VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATION

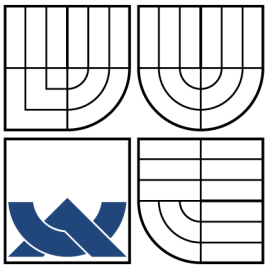
APLIKACE PRO ZABEZPEČENÍ LINUXOVÉHO SERVERU POMOCÍ TECHNOLOGIE SELINUX

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

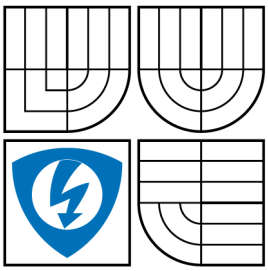
AUTOR PRÁCE
AUTHOR

BC. MICHAL JIRKA

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATION

APLIKACE PRO ZABEZPEČENÍ LINUXOVÉHO SERVERU POMOCÍ TECHNOLOGIE SELINUX

SELINUX APPLICATION FOR LINUX SERVER SECURITY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — BC. MICHAL JIRKA
AUTHOR

VEDOUCÍ PRÁCE — ING. MILAN ŠIMEK
SUPERVISOR

BRNO 2008

ZDE VLOŽIT LIST ZADÁNÍ

Z důvodu správného číslování stránek

ZDE VLOŽIT PRVNÍ LIST LICENČNÍ SMOUVY

Z důvodu správného číslování stránek

ZDE VLOŽIT DRUHÝ LIST LICENČNÍ SMOUVY

Z důvodu správného číslování stránek

ABSTRAKT

Práce se zabývá řízením přístupu v operačních systémech GNU/Linux. Nejdříve je srovnáno volitelné a povinné řízení přístupu a probrány základní technologie založené na povinném řízení přístupu. Blíže je zaměřeno na projekt SELinux, u kterého je vysvětlena tvorba nových pravidel pomocí Type Enforcement. V rámci práce je vytvořena aplikace pro vyhodnocení záznamů technologie SELinux a pro psaní nových pravidel.

KLÍČOVÁ SLOVA

Řízení přístupu, DAC, MAC, AppArmor, SELinux, Type Enforcement, doména, typ, bezpečnostní kontext, GNU/Linux

ABSTRACT

This work is engaged in access control mechanism in GNU/Linux operating systems. At first discretionary and mandatory access control are compared and examine basic technologies based on mandatory access control. More closely is focused on project SELinux, whose generation of new rules is explained. Within the thesis is made application for logging evaluation and for writing new Type Enforcement rules.

KEYWORDS

Access Control, DAC, MAC, AppArmor, SELinux, Type Enforcement, domain, type, security context, GNU/Linux

JIRKA, M. *Aplikace pro zabezpečení Linuxového serveru pomocí technologie SELinux*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 85 s. Vedoucí diplomové práce Ing. Milan Šimek.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Aplikace pro zabezpečení Linuxového serveru pomocí technologie SELinux“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Chtěl bych poděkovat vedoucímu diplomové práce Ing. Milanu Šimkovi za pomoc a cenné rady při zpracování této diplomové práce.

V Brně dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

V dnešní době je rozmach sítí velmi rapidní. Stále více počítačů a serverů je připojeno k internetu a odtud hrozí čím dál větší riziko zneužití. Přesto je v praxi význam ochrany dat často velmi podceňován a hodnota dat je doceněna až při jejich neoprávněném zneužití či zničení.

„Bezpečnost lze definovat jako zajištěnost proti hrozbám, minimalizaci rizik a komplex administrativních, technických, logických a fyzických opatření pro prevenci a detekci neautorizovaného využití dat“ [1]. Základním typem ochrany může být síťový firewall, ale pokud se útočníkovi podaří využít otevřených portů a proniknout do počítače, je tady možnost, že nad ním získá neomezenou vládu.

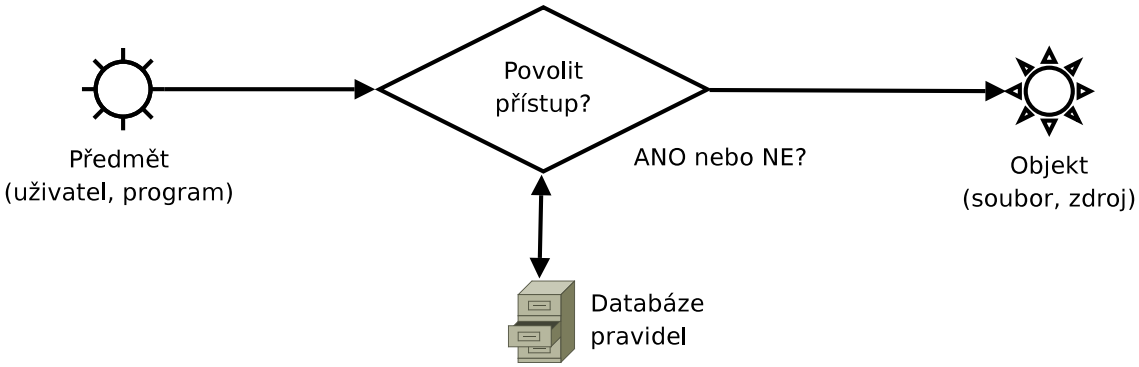
V dnešní době se nejvíce využívá řízení přístupu založené na definici práv pro čtení, zápis a spouštění – tento druh se označuje jako DAC (Discretionary access control – Volitelné řízení přístupu). Systémy Unixového typu však mohou využít nového modelu řízení přístupu, který má označení MAC (Mandatory access control – Povinné řízení přístupu). Jde v podstatě o to, že každý proces má přístup pouze k těm částem systému, které pro svou činnost nezbytně potřebuje. Tím je dosaženo toho, že i když se útočníkovi podaří získat přístup k určitému procesu, tak přesto se mu nepodaří ovládnout celý systém – zůstane jakoby zamčen v části systému.

Na MAC staví základ několik technologií, které jsou vyvíjeny většinou velkými firmami – mezi největší a nejznámější patří systém AppArmor od společnosti Suse a SELinux od firmy RedHat.

1 ŘÍZENÍ PŘÍSTUPU

Dříve nebyl na bezpečnost operačních systémů kladen takový důraz. Jejich zabezpečení bylo na velmi nízké úrovni – uživatel měl například přístup ke všem souborům. Se vzrůstajícím využitím počítačů však vznikla potřeba zavést nějaký systém řízení přístupu k souborům.

Základní princip rozhodování, zda je subjekt (zde se tak označuje uživatel či počítačový program) oprávněn přistupovat k určitému objektu (souboru či prostředku), je ukázán na obr. 1.1. Mezi subjektem a objektem je rozhodovací mechanizmus, který pracuje podle určitých pravidel, která bývají uložena v databázi.



Obr. 1.1: Základní princip řízení

Mechanizmus řízení přístupu je základním a velmi důležitým aspektem bezpečnosti každé aplikace. Jako příklad lze uvést rozsáhlejší webovou aplikaci, která se skládá z uživatelského rozhraní a systému uložení dat. Za použití řízení přístupu lze rozdělit jednotlivé uživatele podle toho, jaká práva budou mít (definuje se co mohou jen číst, k čemu mají neomezená práva nebo také které funkce aplikace jsou oprávněni využívat). Celý systém řízení přístupu má chránit před neautorizovaným prohlížením, změnami a kopírováním jakýchkoliv dat. Tento mechanizmus má také pomoc omezit využití bezpečnostních chyb v programech a jejich následné zneužití útočníkem.

Řízení přístupu k aplikacím a službám se liší v systému řešení. Jsou definovány 2 základní modely řízení přístupu v unixových operačních systémech:

1. DAC (Discretionary access control – Volitelné řízení přístupu)
2. MAC (Mandatory access control – Povinné řízení přístupu)

Na obrázku 1.2 je ukázán pokus o bezpečnostní útok a jeho možné následky na dvou základních modelech. V případě modelu DAC je možné pomocí bezpečnostní

Obr. 1.2: Modely řízení přístupu

chyby systémové komponenty získat kontrolu nad celým systémem. Zatímco u modelu MAC je tento problém ochráněn zvláštními pravidly pro jednotlivé aplikace a pokud jsou pravidla napsána správně, tak by žádná z aplikací neměla mít takový přístup k systému, aby mohl útočník získat plnou kontrolu nad systémem.

Nyní se více zaměřím na jednotlivé modely řízení přístupu.

1.1 Volitelné řízení přístupu (DAC)

DAC je řízení přístupu založené na identitě jednotlivých uživatelů a na členství v určitých skupinách. Povolení přístupu je uživateli uděleno na základě identifikace, kterou má zrovna v době autentizace (jedná se o uživatelské jméno, heslo apod.). V typických systémech založených na DAC je vlastník informace schopen změnit její oprávnění podle vlastní vůle. Nevýhodou DAC je, že administrátor není schopen centrálně spravovat tato oprávnění uložených dat. Řízení přístupu DAC často vykazuje jednu či více z následujících vlastností:

- vlastník dat může přenést vlastnictví na další uživatele – odtud také pojmenování „volitelné“
- uživatel, kterému data patří, může jednoduše určit přístupová práva pro ostatní uživatele (čtení, zápis, spouštění)

Základním nedostatkem DAC je fakt, že nedokáže rozpoznat rozdíl mezi počítačovým programem a uživatelem. S velkým rozmachem internetu tohoto problému začalo využívat stále více škůdců. Dnes je skoro každému, kdo se zajímá o počítače, znám pojem *Trojský kůň*. Jeho první varianta se objevila v 70. letech 20. století [3]. Od něj se vyvinuly další druhy škodlivých programů jako např. počítačové viry či tzv. spyware. Tyto programy fungují na jednoduchém principu, kdy uživatel má

určité oprávnění k objektům na počítači. Jím spouštěné programy tím pádem získávají stejná práva k objektům jako on. Pokud se škodlivý software spustí pod tímto uživatelem, získá samozřejmě úplně stejné možnosti přístupu jako uživatel.

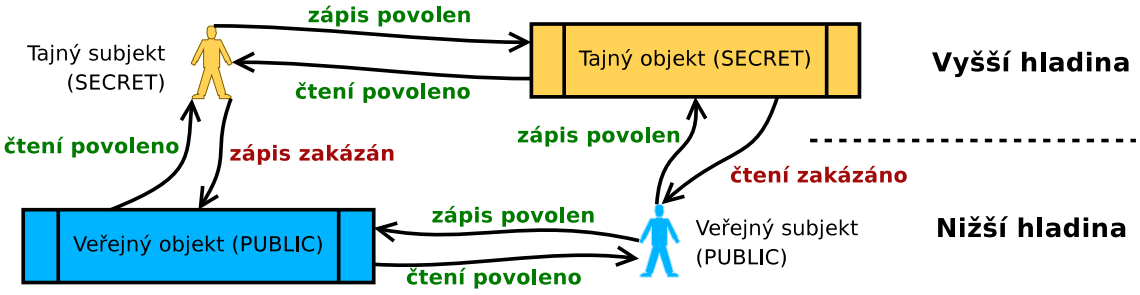
1.2 Povinné řízení přístupu (MAC)

Povinné řízení přístupu zavádí pro každý subjekt práva, která má vůči objektům. Tato jsou specifikována správcem systému jako pravidla, která poté vynucuje operační systém. Pokud se například útočník pokusí dostat do systému přes chybu v httpd démonu, získá pouze kontrolu nad tímto objektem, který má práva omezena pouze na svou činnost. Prostředky, kterými je možno v Linuxu dosáhnout MAC, jsou např. AppArmor či SELinux.

Jako základ povinného řízení přístupu MAC je považován mechanizmus označovaný jako MLS (Multilevel security – Vícevrstvá bezpečnost). Všechny předměty a objekty jsou označeny určitou hladinou bezpečnosti. Na obr. 1.3 jsou ukázány 2 hladiny:

1. public – veřejná
2. secret – tajná

Každá hladina označuje jinou citlivost dat – tajné objekty jsou samozřejmě na vyšší hladině než objekty veřejné. V MLS mohou předměty číst a zapisovat do objektů na stejné hladině, zatímco na vyšší hladinu mohou pouze zapisovat a z nižší hladiny mohou pouze číst. Tato metoda je založena na principu, kdy informace by měla putovat od nižších hladin po vyšší a ne naopak. Tím se teoreticky zvětší ochrana dat na vyšší úrovni.



Obr. 1.3: MLS

Základním nedostatkem MLS je fakt, že realizuje pouze jeden bezpečnostní záměr (tj. ochrana citlivých dat použitím modelu klasifikace důvěrných dokumentů) v přesně definovaném, neměnném stylu.

1.3 Řízení přístupu založené na rolích (RBAC)

Pomocí této metody je omezován přístup k systému pouze pro oprávněné uživatele. Jedná se o nové pojetí řízení přístupu vzhledem k DAC a MAC. Každému subjektu jsou přiřazeny určité role a na základě členství v nich je mu poté povolován či zamezován přístup k souborům či procesům. Tímto není nutné pro každého uživatele definovat oprávnění zvlášť, ale pouze nového uživatele přiřadit již vytvořeným rolím. Velkým problémem tohoto přístupu může být obrovské množství rolí a následně hodně složitá správa. Proto jeho využití není vhodné u systému s velkým množstvím subjektů.

Zde již byl nastíněn typ povinného řízení přístupu, které je použito v SELinuxu. Nyní popišme alternativní technologie zajišťující povinné řízení přístupu – od společnosti Novell systém AppArmor a projekt GrSecurity. Po představení těchto alternativ se již naplno budeme věnovat technologii SELinux.

2 APPARMOR - APLIKAČNÍ BEZPEČNOST PRO LINUX

AppArmor je technologie vyvinutá společností Novell a primárně určená pro operační systém SUSE Linux Enterprise. V dnešní době je tento produkt dostupný i v bezplatné verzi OpenSUSE. Bezpečnost síťových aplikací je zajišťována pomocí MAC. Programy jsou tak chráněny proti zneužití bezpečnostních děr. AppArmor obsahuje vše, co je potřeba pro poskytnutí efektivní ochrany pro běžící aplikace. A to i pro takové, které běží pod správcem systému – v linuxových systémech označovaným jako *root*. Technologie se snaží mařit veškeré pokusy o útok za pomoci bezpečnostních děr a to dokonce i pokusy o zero-day útok[1].

2.1 Základní vlastnosti

AppArmor je dodáván jako Open-source[2], a proto jeho použití nebrání žádné licenční poplatky.

Bezpečnostní díry ve složitých programech jsou lákadlem pro mnohé útočníky, kteří se pokouší za pomoci těchto vad proniknout do napadeného systému a dostat se k citlivým datům uloženým na počítači. Základní bezpečnostní prvky uvnitř sítě řeší pouze část tohoto problému, protože v dnešní době je potřeba mít povolen přístup i uživatelům z vně sítě – jde například o zaměstnance na služebních cestách. Mimoto firewally poskytují pouze velmi malou ochranu proti stále narůstajícímu počtu útoků, které mají původ uvnitř dané podnikové sítě.

AppArmor poskytuje prevenci průniků a snaží se ochraňovat operační systém a aplikace proti škodlivým vlivům zevnitř i zvnějšku počítačové sítě – jedná se o útoky, škodlivé aplikace a v neposlední řadě o počítačové viry.

Společnost Novell uvádí tyto pozitivní vlastnosti svého produktu:

Žádné trhliny v technologii – AppArmor ochraňuje data proti neoprávněným uživatelům a ochraňuje systém také před aplikacemi, do kterých bylo pomocí bezpečnostních děr již proniknuto.

Jednoduché užití – není potřeba žádné přeznačení souborového systému a v základní konfiguraci je spousta předdefinovaných profilů pro nejběžnější síťové služby – jako například webový, e-mailový či ssh server.

[1] Zero-day útoky využívají zranitelnost kódu aplikací, u kterých dodavatel ještě neuvolnil příslušné bezpečnostní záplaty.

[2] Open-source software (OSS) jsou aplikace s otevřeným zdrojovým kódem. Otevřenost značí jak fyzickou dostupnost kódu, tak především legální dostupnost.

Precizní kontrola – je poskytována schopnost omezovat jednotlivé internetové aplikace i v případě, že jsou spouštěny uvnitř serveru Apache a při využití jeho modulů. AppArmor poskytuje serveru Apache možnost změnit kontext a aplikovat specifické bezpečnostní profily na jednotlivé skripty před jejich spuštěním.

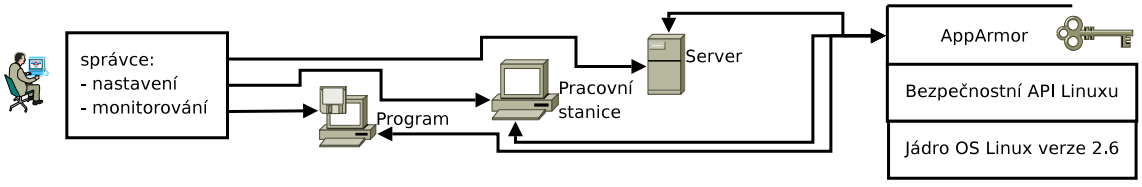
Zamezení problémů – zvyšující se počet tzv. zero-day útoků má za následek, že bezpečnostní díry jsou často objeveny dříve než je na ně aplikována bezpečnostní záplata. AppArmor se pokouší zastavit útoky ještě dříve, než se jim podaří proniknout do systému, a tím omezit toto rychlé aplikování záplat.

2.2 Technické řešení

Bezpečnostní model využívá systému tzv. bílých listin[3], avšak není použita žádná databáze označení útoků. Jednotlivé profily povolují přístup pouze k nezbytným adresářům a souborům a kompletní přístup k jádru je pouze přes bezpečnostní API systému Linux (Linux Security Module).

Spousta činností je již zautomatizována:

- *automatické prohledávání*: hledání programů naslouchajících na otevřených portech a kontrola již existujících profilů pro ně
- *automatické generování*: vytvoření šablony profilu na základě sledování činnosti aplikace
- *automaticky učící se režim*: automatické rozšiřování již existujícího profilu během činnosti sledované aplikace
- *interaktivní optimalizátor*: navrhování nejlepších pravidel a asistence při zjednodušování již existujících profilů



Obr. 2.1: Technické řešení technologie AppArmor

[3]Programy či služby na bílé listině mají povolen přístup.

3 PROJEKT GRSECURITY

Grsecurity je sada záplat linuxového jádra s důrazem na vylepšení bezpečnosti systému. Jeho typické nasazení je na webových serverech a systémech s velkým počtem vzdálených přístupů z nedůvěryhodných oblastí.

Příklad nastavení v konfiguračním souboru grsecurity:

```
/usr/bin/appl o {
   /etc/appl.conf r

   connect {
      147.229.187.111:80 stream dgram tcp udp
   }
}
```

Výpis výše povolí programu *appl* čtení konfiguračního souboru a připojení na danou adresu a daný port uvedenými protokoly.

3.1 PaX

PaX je hlavní komponentou v balíku utilit GrSecurity. Ta přináší podporu nespustitelných stránek na architektuře x86. Jiné architektury, na rozdíl od x86 umožňují přímé označování stránek za nespustitelné. Na x86 to není možné, protože v tabulce stránek je pouze 1 bit specifikující, zda je možno na danou stránku zapisovat. PaX implementuje nespustitelné stránky 2 způsoby:

1. Využití rozdělení virtuálního adresního prostoru na tzv. segmenty. Deskriptory daných segmentů umožňují nastavit, zda je daný segment spustitelný, čitelný či zapisovatelný.

2. Využívá se faktu, že dnešní CPU mají 2 TLB jednotky (Translation Lookaside Buffer) – jednu pro data a druhou pro instrukce. TLB je vyrovnávací paměť, kam jsou procesorem ukládány výsledky překladu adres při stránkování paměti, aby nemusel při každém přístupu do ní procházet obsah stránkovacích struktur. PaX u všech stránek, které mají byt nespustitelné, nastaví tzv. supervizor bit. To způsobí, že při přístupu na tuto stránku z kódu s CPL3 (Current Privillege Level, CPL3 značí uživatelské aplikace) dojde k vyvolání výjimky procesoru. V této výjimce se provede zjištění, zda šlo o provedení instrukce či datový přístup – v případě pokusu o spuštění kódu se aplikace ukončí. Pokud se jedná o přístup z jádra (CPL nižší než 3), není vyvolána žádná výjimka.

4 SELINUX

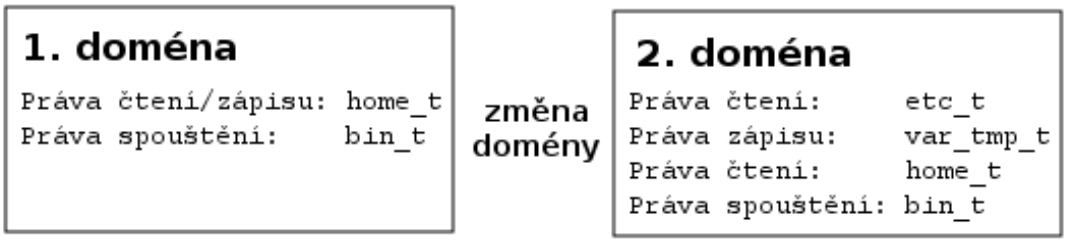
SELinux je systém obsahující mnoho mechanizmů, které dokáží ochránit před útoky využívajícími bezpečnostní díry v aplikacích. Obsažena je i ochrana proti zero-day útokům. SELinux konkrétně využívá RBAC a sandboxing[1]. Technologie dále poskytuje nástroje pro monitorování a vyhodnocení pokusů o překročení definovaných práv. Správce je schopen monitorováním systémových záznamů objevit pokus o prolomení práv a poté znemožnit vetřelci nežádoucí činnost uvnitř systému.

SELinux je navržen pro ochranu například proti:

- neoprávněnému čtení dat
- neoprávněné modifikaci dat
- obcházení bezpečnostních mechanizmů aplikací
- narušení pomocí ostatních procesů v systému

4.1 Princip činnosti SELinuxu

SELinux přiřadí každé aplikaci či procesu určitou doménu. Jednotlivé domény mají přiřazenu množinu postačujících oprávnění k tomu, aby mohly bezchybně pracovat. Například v doméně je určeno, ke kterým souborům má přístup a jaké typy operací může provádět s těmito soubory. Aby toto bylo možné, je potřeba, aby každý soubor měl nějaké označení – tomu se zde říká bezpečnostní kontext (*security context*). Doména je definována tak, že je určeno jaké operace je možno provádět se soubory s určitým bezpečnostním kontextem. Nemá přístup k souborům, které mají jiný kontext než takový, ke kterému má povolen přístup.



Obr. 4.1: Princip SELinuxu

[1] uzavření aplikace v zabezpečeném prostředí

Velmi zjednodušený model principu, na kterém SELinux funguje, je naznačen na obr. 4.1. Proces spouštějící program opustí dosavadní doménu a provede změnu na novou, jež může mít více či méně oprávnění. Tady programy mohou spouštět další aplikace s většími či menšími právy než samy mají.

Základní nástroj SELinuxu známý jako *type enforcement* (TE) zajišťuje jednoduchou čitelnost pravidel definujících domény. SELinux obsahuje také druhotnou ochranu založenou na RBAC, která omezuje přístup uživatelů do jednotlivých domén – to je výhodné pokud chceme některé domény přístupné jen správci systému. Definice domén, bezpečnostních kontextů a změn mezi doménami se objevují v souborech s definovanou politikou (v tzv. policy files). Tyto soubory může měnit pouze správce SELinuxu.

4.2 Bezpečnostní model

Bezpečnostní model SELinuxu zahrnuje 3 základní prvky:

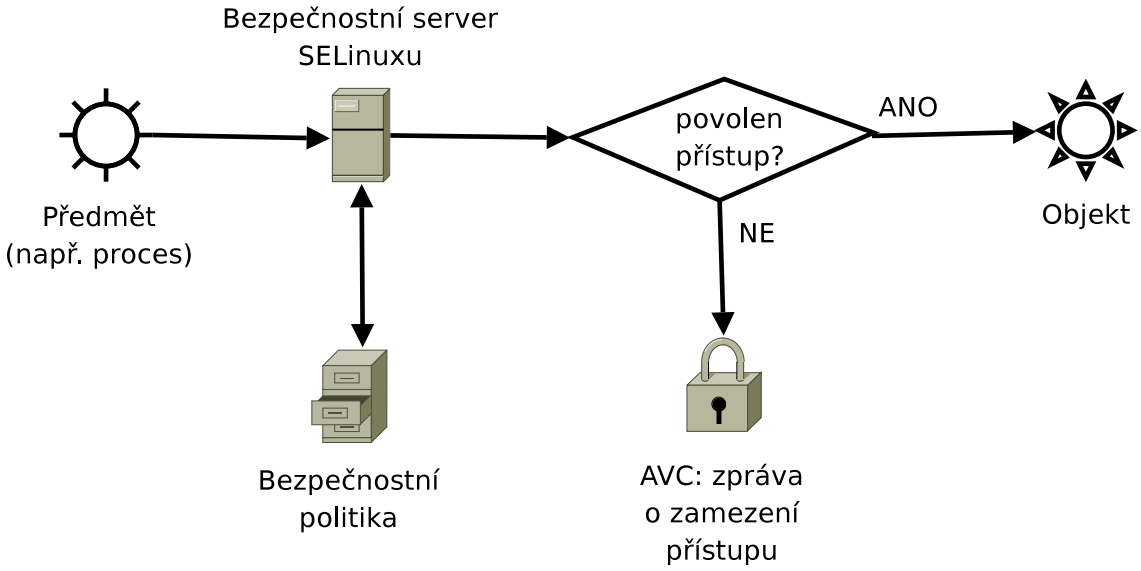
1. subjekt – typy procesů
2. objekt – typy objektů, ke kterým předmět přistupuje
3. akce – činnost, která je povolena

SELinux definuje velmi mnoho bezpečnostních tříd objektů (všechny jsou k nalezení například v [2]). Za zmínku stojí adresáře, soubory, souborové systémy, spoje, procesy, speciální soubory různých typů (bloková zařízení, sokety, FIFO, ...). Procesy mohou sloužit jako subjekty, ale také jako objekty.

V Linuxu je většina entit reprezentována soubory (adresáře, zařízení a také paměť může být přístupná pomocí souborů), a proto jsou nejpoužívanější třídou objektů právě soubory.

Akce, které může SELinux vykonávat nad objekty se liší na základě jeho typu. Například nad objektem souborového typu mohou být vykonávány akce jako připojení (Append), vytvoření (Create), spuštění (Execute), získání vlastností (Get attribute), správa vstupu/výstupu (I/O control), spojení (Link), zamknutí (Lock), čtení (Read), přejmenování (Rename), rozpojení (Unlink), zápis (Write).

Využitím této jednoduché struktury (předmět – akce – objekt) můžeme lehce rozhodnout, zda daný předmět je oprávněn k vykonání dané akce na tomto určitém objektu. Bezpečnostní rozhodnutí jsou uložena v databázi, a proto SELinux dosahuje vysokého stupně flexibility. Základní rozhodování je naznačeno na obr. 4.2.



Obr. 4.2: Proces rozhodování u SELinuxu

Bezpečnostní kontexty

V každém moderním operačním systému je řízení přístupu založeno na nějakém typu kontroly bezpečnosti z hlediska subjektu a objektu. V SELinuxu se používá vlastnost zvaná *bezpečnostní kontext (security context)*. Všechny objekty (soubory, komunikační kanály, sokety, síťové prvky atd.) a subjekty (uživatelé, procesy) mají přiřazen jedinečný bezpečnostní kontext. Tento kontext se skládá ze 3 prvků:

- **uživatel (user)** - jedná se o uživatelský účet SELinuxu, který je spojen s určitým subjektem nebo objektem. V případě subjektu se jedná o uživatele SELinuxu, pod kterým daný proces poběží. Pokud se jedná o objekt, jde o uživatelský účet, který vlastní daný objekt.

- **role (role)** - v SELinuxu mohou uživatelé vstoupit do různých rolí (každá z nich definuje jiná oprávnění přístupu). V jeden čas je však uživateli umožněno členství pouze v jedné z rolí, avšak každému uživateli je povoleno přejít do role jiné použitím speciálního příkazu `newrole` (jedná se o něco podobného, jako když se použije v Linuxu příkaz `su` pro získání superuživatelských práv). Hlavní rolí v SELinuxu je *sysadm_r*, která se používá pro správce systému.

- **typ (type)** - označovaný také jako doména, rozděluje subjekty a objekty do skupin, které spolu nějakým způsobem souvisí. Typy jsou základním bezpečnostním prvkem SELinuxu a je podle nich rozhodováno o autorizaci. Pomocí nich jsou vytvářeny *sandboxy*, do kterých jsou běžící procesy jakoby uzavřeny, a nemohou tak získat větší oprávnění.

Obvyklý formát pro zápis bezpečnostních kontextů má tento tvar:

```
uživatel:role:typ
```

V tab. 4.1 je vidět konvence při pojmenování uživatelů, rolí a typů.

Tab. 4.1: Konvence pojmenování bezpečnostních atributů

Atribut	**Přípona jména**	**Příklad pojmenování**
Uživatel	–	root
Role	_r	sysadm_r
Typ	_t	sysadm_t

SELinux modifikuje některé systémové příkazy – je jim přidána volba `-Z`, která zajišťuje zobrazení bezpečnostních kontextů objektů a subjektů. Například příkaz `ls -Z` zobrazí výpis aktuálního adresáře s viditelnými bezpečnostními kontexty souborů a adresářů. Velmi použitelný je příkaz `id -Z`, který zobrazí uživatelský kontext. Výpis může vypadat podobně tomuto:

```
# id -Z
jirkami:user_r:user_t
```

V tab. 4.2 je srovnání řízení přístupu ve standardním Linuxovém operačním systému a při použití technologie SELinux. V klasických operačních systémech se používá známý systém nastavení tří bitů – pro čtení/zápis/spuštění. Tato oprávnění se definují pro vlastníka souboru, skupinu vlastníka a pro všechny ostatní. V SELinuxu jsou atributy řízení přístupu vždy bezpečnostní kontexty.

Tab. 4.2: Srovnání řízení přístupu v Linuxu a s použitím technologie SELinux

	Standardní Linux	**S použitím SELinuxu**
Bezpečnostní atributy procesů	Skutečné a efektivní ID uživatele či skupiny	Bezpečnostní kontext
Bezpečnostní atributy objektů	Režimy přístupu a ID majitele souboru či skupiny	Bezpečnostní kontext
Základ pro řízení přístupu	ID uživatele procesu či skupiny a režim přístupu založený na ID majitele souboru či skupiny	Oprávnění povolena mezi typem procesu a typem souboru

4.3 Základní politika SELinuxu

Základní cestou k vytvoření souboru s bezpečnostní politikou je jeho sestavení ze zdrojového souboru se zásadami bezpečnosti (`policy.conf`). K tomu se využívá nástroj `checkpolicy`, který při použití zkontroluje zdrojový soubor na správnost syntaxe a sémantiku zápisu jednotlivých pravidel. Pokud je vše v pořádku, je výsledek zapsán do tzv. binární formy, která je čitelná pomocí jádra SELinuxu.

Mezi základní části zdrojového souboru patří:

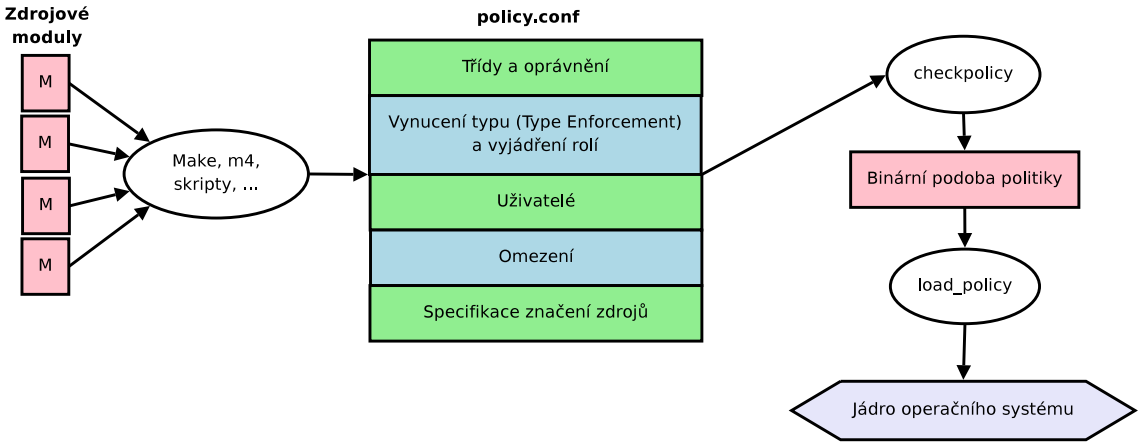
- **Třídy a oprávnění** – definice tříd objektů a oprávnění pro jednotlivé třídy. Správce SELinuxu by nikdy neměl měnit tyto vlastnosti.
- **Vynucení typu (Type Enforcement) a vyjádření rolí** – tato část obsahuje definici TE pravidel – jedná se o největší díl politiky SELinuxu. Najdeme zde deklarace typů a pravidel TE (tzn. všechna `allow`, `type_transition` a další předpisy). Blíže se Type Enforcement budeme věnovat v kapitole 4.3.1. Tato část také obsahuje definice rolí blíže popsané v 4.3.2.
- **Uživatelé** – tato kapitola je věnována definici uživatelů. Vše je popsáno více v části 4.3.3.
- **Omezení** – zde jsou poskytovány prostředky pro omezování TE politiky navzdory tomu, co je povoleno pomocí pravidel Type Enforcement. Této části je více věnováno v kapitole 4.3.4.
- **Specifikace značení** – všechny objekty musí byt označeny bezpečnostním kontextem SELinuxu pro vynucení řízení přístupu. Tato sekce ukazuje, jak zacházet se systémem souborů za účelem značení, a obsahuje pravidla pro označování dočasných objektů, které vznikají během běhu systému. Tyto věci jsou přiblíženy v části 4.3.5.

Běžně se dnes používá tzv. monolitická politika SELinuxu – zásady jsou sestaveny jako jeden binární soubor pomocí nástroje `checkpolicy` a tento soubor je okamžitě načten do jádra. Ale vzhledem k velké obsáhlosti a rozlehlosti politiky SELinuxu se využívá „skládání“ z menších částí – z modulů. Zdrojové moduly jsou kombinací textových souborů, shellových skriptů, m4 maker a Makefile souborů. Spojováním se vytváří jeden velký soubor `policy.conf`, který je poté sestavován pomocí `checkpolicy` v binární podobu čitelnou jádrem operačního systému.

Konstrukce binárního souboru z jednotlivých modulů je naznačena na obrázku 4.3 – jednotlivé moduly jsou nejdříve pomocí různých skriptů a maker skládány do jednoho souboru `policy.conf`. Následuje jejich kompilace do binární podoby za pomoci `checkpolicy` a následné načtení do jádra užitím nástroje `load_policy`.

Celý tento proces se může zdát náročný, ale opak je pravdou. Vše je totiž zautomatizováno pomocí souborů `Makefile`, které jsou v každém adresáři se zdrojovým modulem. Samotná kompilace tedy probíhá pomocí standardního příkazu `make` s těmito možnými cíly:

- `policy` – vytvoření `policy.conf` a jeho kontrola
- `install` – shodné s `make policy` + nainstalování binární podoby tak, aby mohla být načtena jádrem při dalším startu systému
- `load` – udělá totéž jako `make install` a ihned načte binární podobu pravidel do jádra. Tato politika bude okamžitě funkční.
- `reload` – provede stejné činnosti jako `make install` a znovu načte binární pravidla do jádra
- `relabel` – přeznačení souborového systému s ohledem na definici kontextů



Obr. 4.3: Sestavení a načtení politiky SELinuxu ze zdrojových modulů

Tak například `make install` udělá všechny kroky naznačené na obr. 4.3 mimo posledního (načtení do jádra).

4.3.1 Type Enforcement - Vynucení typu

SELinux Type Enforcement asociuje každý proces s určitou doménou a každý objekt jiný než *proc* s typem. Povolení definují operace, které mohou být s objekty vykonány. Například proces webového serveru Apache běží v doméně `httpd_t`, a proto vlastní oprávnění přiřazená této doméně. Politika SELinuxu dává oprávnění doménám a specifikuje pravidla pro přechody mezi nimi.

Oprávnění jsou označována jako přístupové vektory, jež specifikují operace, které může doména provádět s objekty daného typu (jako například se soubory). Na příkladě serveru Apache – pravidla přístupových vektorů umožňují procesům běžícím v doméně `httpd_t` zápis do logů webového serveru.

Pod Linuxem může každý program vytvářet nové podprocesy. Tyto mohou běžet ve stejné doméně jako rodičovský proces nebo může být specifikována nová doména, do které bude nově spouštěný podproces přiřazen. Programy, které umožňují vstup do nové domény po spuštění, se nazývají vstupními body do domény. Příkladem může být proces `init` běžící v doméně `initrc_t`. Pokud však tento proces pustí například webový server, ten automaticky přejde do domény `httpd_t` (přesně jak je definováno v pravidlech SELinuxu).

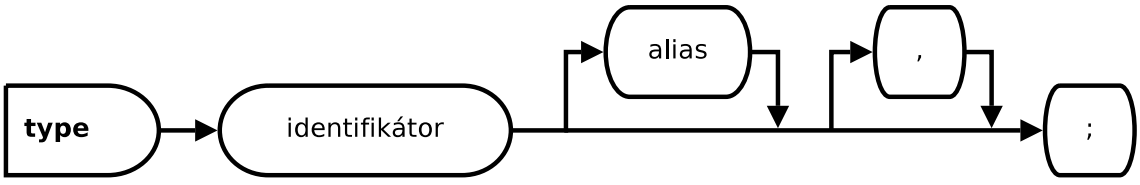
Deklarace TE se skládá ze sedmi různých částí:

1. **attribute_def** – deklarace atributů
2. **type_def** – deklarace typů
3. **typealias_def** – definice alternativních názvů typů
4. **bool_def** – deklarace logických proměnných
5. **transition_def** – specifikování přechodů mezi doménami
6. **te_avtab_def** – nastavování tabulky přístupových vektorů TE
7. **cond_stmt_def** – definice podmíněných tvrzení

Deklarace typů

Tvorba zásad SElinuxu vyžaduje, aby byla všechna jména typů explicitně definována. Jednoduchá forma deklarace je

```
type ping_t;
```



Obr. 4.4: Deklarace typu (type_def)

Pro jeden typ může být definováno více alternativních názvů (tzv. aliasů). Celá definice je naznačena na obrázku 4.4. Za klíčovým slovem `type` následuje identifikátor typu a případně seznam aliasů oddělený čárkou.

Příkladem takového zápisu může být deklarace dvou typů v souboru *ping.te*:

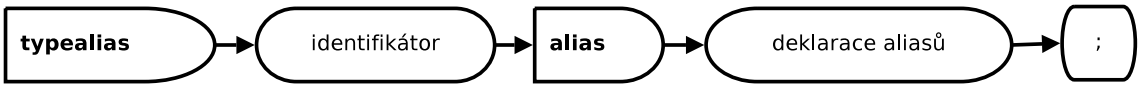
```
type ping_t, domain, privlog;
type ping_exec_t, file_type, sysadmfile, exec_type;
```

Zde je deklarováno jméno typu `ping_t`, atributy `domain` a `privlog` jsou s ním asociovány. Druhý řádek deklaruje typ se jménem `ping_exec_t` a sdružuje s ním atributy `file_type`, `sysadmfile` a `exec_type`.

Alternativní názvy typů (aliasy)

Pro definici alternativních názvů typů se také využívá speciálně určená deklarace, která je naznačena na obr. 4.5. Příkladem takového zápisu může být například definice v souboru *cups.te*:

```
typealias cupsd_etc_t alias etc_cupsd_t;
```



Obr. 4.5: Deklarace alternativního názvu typu

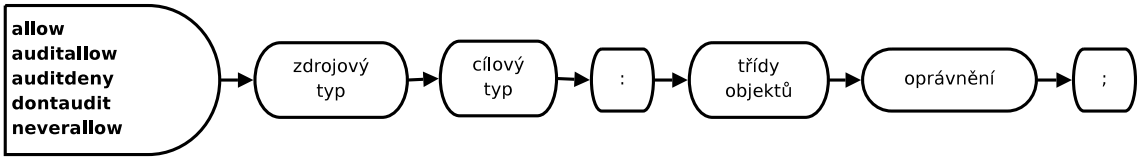
Společné atributy typů

Některé typy využívají stejnou vlastnost, a proto se definuje tzv. atribut typu, který umožňuje „svázání“ jednoho či více typů a tak definovat jejich skupinu se stejnými vlastnostmi. Příkladem je definování atributu, jenž opravňuje ke čtení ze záznamů systému. Tyto společné názvy atributů jsou definovány v souboru `attrib.te` – soupis nacházející se v systému Fedora Core je možné najít v [2] (příloha E).

Ukázka deklarace společného atributu se jménem `admin`:

```
attrib admin;
```

Přístupové vektory TE



Obr. 4.6: Schéma definice přístupového vektoru

Veškerá povolení bezpečnostním systémem SELinux jsou uložena v datovém prostoru jádra – v objektu, který je nazýván přístupovou maticí TE (TE access matrix).

Jsou definovány 4 základní činnosti, kterým se v SELinuxu říká vektory:

allow – operace povolené, ale bez zaznamenávání

auditallow – činnosti s povolením a takové, které se zapíší do záznamů, pokud nastanou

auditdeny – operace, které jsou zablokovány a zaznamenány při výskytu

dontaudit – zakázané činnosti a také není pořizován žádný záznam při jejich pokusu o přístup

Na obrázku 4.6 je schématicky naznačeno psaní pravidla pro přístupový vektor. Nejdříve se uvede klíčové slovo označující jednu z pěti činností, která je provedena s daným typem. Čtyři z nich jsou popsány výše, pátá je `neverallow` – ta definuje zamezení přístupu, které se musí SELinuxem sledovat. Jde o volitelnou vlastnost. Je sice považováno za bezpečné vektory s `neverallow` pravidlem uvádět, ale není to nutné, protože primárně je vše zakázáno. Zápis jednoho pravidla má tyto další části:

- **zdrojový typ** – obvykle je to typ asociovaný s procesem, který se pokouší o přístup
- **cílový typ** – typ objektu, který má být zpřístupněn procesem
- **třídy objektů** – třída objektu, která má tento specifický přístup povolen
- **oprávnění** – druh přístupu, který je zdrojovému procesu povolen k cílovému typu pro uvedené třídy objektů

Zde je čas ukázat si zápis přístupového vektoru (povolení deklarace asociované s typem `ping_t`):

```
allow ping_t ping_exec_t:file { read getattr lock execute ioctl };
```

Pravidlo, vytvořené touto deklarací, povoluje procesu bežícímu s typem `ping_t` pracovat se souborem majícím typ `ping_exec_t` – a to číst, získávat vlastnosti, uzamknout, spustit a kontrolovat vstupní a výstupní činnosti.

Speciální typy zápisu V tabulce 4.3 je ukázka speciální notace, kterou je možno použít při psaní přístupových vektorů TE.

Tab. 4.3: Speciální notace pro popis typů, tříd a oprávnění

Notace	Popis
*	všechny prvky
∼	komplementace; doplněk k zadanému
–	odečtení; zakázané prvky
self	cílový typ stejný jako zdrojový

Makra specifikující a povolující přechody Aby byl povolen přechod mezi doménami, musí být povolena jak změna typu, tak také změna přístupového vektoru. Psaní povolení pro obě části zároveň nám zjednodušují některá M4 makra, která vhodně generují přechody typů a přístupových vektorů z jednořádkového zápisu. Nejpoužívanějšími makry jsou:

`domain_auto_trans` – specifikuje a povoluje přechod související se spouštěním programu definovaného jako vstupní bod domény

`file_type_auto_trans` – stanovuje a opravňuje k přechodu, který souvisí s vytvářením souborů

Nyní si uveďme příklady obou maker:

```
domain_auto_trans(initrc_t, ping_exec_t, ping_t);
file_type_auto_trans(ftpd_t, var_log_t, xferlog_t, file);
```

První řádek našeho příkladu nám vygeneruje tato pravidla (nejdříve je přechod z domény `initrc_t` na `ping_t` v případě, že je spuštěn proces s typem `ping_exec_t`, a následně je tento přechod povolen):

```
type_transition initrc_t ping_exec_t:process ping_t;
allow initrc_t ping_t:process transition;
```

A při použití `file_type_auto_trans`:

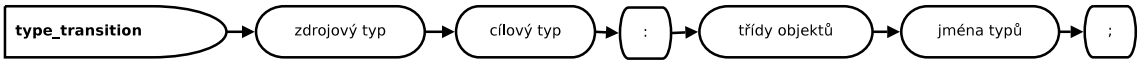
```
type_transition ftpd_t var_log_t:file xferlog_t;
allow ftpd_t var_log_t:dir rw_dir_perms;
allow ftpd_t var_log_t:file create_file_perms;
```

Pomocí tohoto jednořádkového makra budou vykonány tyto činnosti:

- Pokud proces `ftpd_t` vytvoří soubor v adresáři označeném `var_log_t`, bude mít tento soubor typ `xferlog_t`.

- Každý `ftpd_t` proces může číst a zapisovat do `var_log_t` adresáře (zápis rw_dir_perms značí tyto akce: read, getattr, lock, search, ioctl, add_name, remove_name, write).
- Jakýkoliv proces označený `ftpd_t` má povolení vytvářet soubory v adresáři s typem `var_log_t` (create_file_perms = create, ioctl, read, getattr, lock, write, setattr, append, link, unlink, rename).

Deklarace přechodů



Obr. 4.7: Definice přechodů mezi typy

Jako první je ukázka změny typu příslušející procesu:

```
type_transition sysadm_t ping_exec_t:process ping_t;
```

Toto nastane, pokud se proces v doméně `sysadm_t` pokusí spustit program mající typ `ping_exec_t`. Tímto se proces pokusí o přechod do domény `ping_t` (přechod musí však být povolen i přístupovým vektorem).

Další může být ukázka přechodu mezi typy, která se týká souboru:

```
type_transition httpd_t var_log_t:file httpd_log_t;
```

Uvedené pravidlo ovlivňuje chování procesu v doméne `httpd_t`, který vytvoří soubor v adresáři označeném `var_log_t`. Nově vytvořené soubory budou mít typ `httpd_log_t`.

I zde existuje několik M4 maker, která ulehčují psaní pravidel. V tabulce 4.4 je uvedeno 8 vhodných, které nám specifikují, kterých tříd objektů se přechod týká.

Definice logických proměnných

Booleovské proměnné nabývají hodnot true/false (pravda/nepravda; logická 1/logická 0) a na základě toho může být pomocí bezpečnostní politiky toto pravidlo testováno. Zápis je velmi jednoduchý – například definice nové proměnné `user_ping` označené logickou nulou je

```
bool user_ping false;
```

Tato booleovská proměnná kontroluje přístup neprivilegovaných uživatelů k příkazům `ping` a `traceroute`. Kontrola této proměnné je využita při podmíněných deklaracích obsažených v souborech `ping.te` a `traceroute.te` – toto bude vysvětleno v další části, která se věnuje definici podmínek.

Tab. 4.4: Makra pro popis tříd objektů

Makro	Definice	Popis
devfile_class_set	{chr_file blk_file}	třídy blokových zařízení
dgram_socket_class_set	{udp_socket unix_dgram_socket}	třídy datagramových soketů
dir_file_class_set	{dir file lnk_file sock_file fifo_file chr_file blk_file}	třídy souborů a adresářů
file_class_set	{file lnk_file sock_file fifo_file chr_file blk_file}	třídy souborů
notdevfile_class_set	{file lnk_file sock_file}	třídy souborů bez blokových zařízení
socket_class_set	{tcp_socket udp_socket rawip_socket netlink_socket packet_socket unix_stream_socket unix_dgram_socket}	třídy soketů
stream_socket_class_set	{tcp_socket unix_stream_socket}	třídy průchozích soketů
unpriv_socket_class_set	{tcp_socket udp_socket unix_stream_socket unix_dgram_socket}	třídy neprivilegovaných soketů kromě třídy IP soketu

Podmíněné definice

Při definici podmíněných příkazů je možno využít kterýkoliv ze šesti operátorů uvedených v tabulce 4.5.

Tab. 4.5: Pomocné operátory při definici podmínek

Symbol	Popis
`&&`	logické AND
`==`	logická rovnost
`!`	logický zápor
`!=`	logická nerovnost
`||`	logické OR
`^`	logické vylučující OR

Samozřejmě nesmí chybět ukázka psaní takovéhoto podmíněného příkazu (tento kód je převzat ze souboru `ping.te`):

```
if (user_ping) {
   domain_auto_trans(unpriv_userdomain, ping_exec_t, ping_t)
   # povolit přístup k~terminálu
   allow ping_t { ttyfile ptyfile }:chr_file rw_file_perms;
   ifdef('gnome-pty-helper.te', 'allow ping_t gphdomain:fd use;')
}
```

Tento podmíněný výraz se rozhoduje, zda jsou neprivilegovaní uživatelé oprávněni využívat příkazy `ping` a `traceroute`. Pouze pokud má proměnná `user_ping` hodnotu `true`, vykonají se příkazy uvedené v podmínce:

- Autorizovat a automaticky přejít z domény označené `unpriv_userdomain` na `ping_t` po vykonání programu označeného typem `ping_exec_t`.

- Povolit doméně `ping_t` přístup k uživatelským službám terminálu (TTY nebo PTY).
- Vyvolat M4 makro `ifdef`, které podmíněně povolí doméně `ping_t` využívat vlastnosti souborů pocházejících z domény `gphdomain`.

4.3.2 Role

SELinux poskytuje formu RBAC postavenou nad Type Enforcement (TE). Role se využívají k seskupování doménových typů a k omezení vztahů mezi uživateli a doménovými typy. Uživatelé v SELinuxu sdružují jednu nebo více rolí s linuxovým uživatelem. Užití RBAC umožňuje efektivní definici a správu oprávnění.

SELinux nemá žádné vestavěné role – vyjímku tvoří pouze `object_r`. Role jsou stejně jako typy definovány v pravidlech a dostávají význam během užívání. V definici rolí se využívají tyto čtyři body:

1. deklarace rolí
2. povolování užitím `allow`
3. změny mezi rolemi
4. nadřazenost rolí

Deklarace rolí

Role se definují podle následující syntaxe:

```
role název_role [types přiřazené_typy]
```

Význam tohoto zápisu:

- **název_role** – jmenný identifikátor role. Může být libovolně dlouhý a skládá se z ASCII znaků, čísel a podtržítek (_).
- **přiřazené_typy** – jeden nebo více identifikátorů typů. Více typů se zadává jako seznam oddělený mezerami ve složených závorkách (např. { user_t passwd_t }). Může být také použita syntaxe se znaménkem mínus před názvem typu – tento typ bude vynechán.

Role také může být definována bez přiřazení typů – v takovém případě se píše pouze klíčové slovo `role` a její název.

Jedné roli může být přiřazeno současně několik typů. Definice může být uvedena na jednom řádku s typy uvedenými ve složených závorkách nebo je možné použít definici role vícekrát, což je vidět na ukázce:

```
role user_r types user_t;
role user_r types passwd_t;
```

Povolení užitím `allow`

Pomocí klíčového slova `allow` je definováno, které role je možné mezi sebou měnit. Úspěšná změna role je také závislá na tom, zda je uživatel oprávněn k užití role, do které chce přejít. Zápis povolení přechodu mezi rolemi je následující:

```
allow staff_r sysadm_r;
```

Díky tomuto je umožněno procesu běžícímu pod rolí `staff_r` změnit na roli `sysadm_r` během změny domény. Je však umožněn přechod v dopředném směru (`staff_r` → `sysadm_r`), pro zpětný přechod je nutné nadefinovat nové povolení s opačným pořadím názvů rolí.

Teď již je čas vysvětlit si základní syntaxi povolování přechodů mezi rolemi:

`allow` *`množina_rolí množina_rolí`* `;`

Množina_rolí – jeden nebo více identifikátorů rolí. V případě zápisu více rolí dohromady se používá již známá notace = seznam oddělený mezerami uzavřený ve složených závorkách.

Změny mezi rolemi

Role se stejně jako typy mohou během práce programu měnit. Proto je dobré tuto činnost co nejvíce zautomatizovat. Pro typy se využívá klíčové slovo `type_transition`, které zajišťuje automatické přechody mezi nimi. U rolí se logicky používá zápis uvozený `role_transition` a je velmi podobný jako při změně typů. Ukázkou je tento zápis:

```
role_transition sysadm_r http_exec_t system_r;
```

Význam by již měl jít ze získaných znalostí odhadnout, ale určitě bude lepší ho vysvětlit – pokud proces s rolí `sysadm_r` spustí soubor mající typ `http_exec_t`, pokusí se SELinux změnit roli na `system_r`.

Základní syntaxe zápisu:

`role_transition` *`množina_rolí množina_typů role`* `;`

- **množina_rolí** – jeden nebo případně více identifikátorů rolí uvedených ve složených závorkách
- **množina_typů** – názvy typů, které obsahuje spouštěný soubor

- **role** – nová role po úspěšném přechodu

Stejně jako u změny typu i zde musí být přechod uveden v povolovacích zápisech pomocí `allow`.

Nadřazenost rolí

Pomocí klíčového slova `dominance` se vytváří hierarchická vazba mezi rolemi, kdy dominantní role dědí všechny asociace typů od rolí, kterým dominuje.

Syntaxe je následující:

`dominance { role` *`jméno_role`* `{` *`množina_rolí`* `} }`

- **jméno_role** – identifikace role libovolné délky složená z ASCII znaků, čísel a podtržítek
- **množina_rolí** – jedna nebo více rolí specifikovaných ve tvaru `role` *`jméno_role`*. Při výčtu více rolí se oddělují středníkem a uvádí do složených závorek.

Pro ukázku můžeme uvažovat následující tvar zápisu:

```
dominance { role super_r {role sysadm_r; role secadm_r;} }
```

Zde je vidět, že `super_r` bude mít všechny typy asociované sobě i rolím `sysadm_r` a `secadm_r`.

4.3.3 Uživatelé

Nyní se zaměřme na definici uživatelů. Nejprve je dobré si připomenout, že linuxový a SELinux uživatel jsou naprosto odlišní a často nesouvisející. Například u jednoho procesu se může lišit identifikátor linuxového a uživatele SELinuxu.

Syntaxe je opět poměrně standardní – nadefinuje se jméno uživatele a přiřadí se mu jedna nebo více rolí:

`user` *`uživatelské_jméno`* `roles` *`množina_rolí`* `;`

- *uživatelské_jméno* – jmenná identifikace uživatele libovolné délky složená z ASCII znaků, čísel a podtržítek.
- *množina_rolí* – jeden či více identifikátorů rolí, které již musí být nadefinovány dříve v bezpečnostních zásadách. Pokud je typů více, oddělují se mezerou a uzavírají do složených závorek.

Ukázka jednoduchého přiřazení role uživateli:

```
user george roles { user_r };
```

V tomto příkladě je definován SELinux uživatel se jménem `george` a je přiřazen do role `user_r`. Definice uživatelů má přesné místo v souboru s bezpečnostní politikou – uvádí se za definicí všech typů a rolí a před určení omezení. U uživatelů neexistují žádné přechody ani povolení pomocí `allow` – je to z důvodu původního záměru, kdy by měl být uživatel neměnný.

„Mapování“ linuxového uživatele na uživatele SELinuxu

Programy spouštěné během přihlášení (jako například `login` či `sshd`) jsou zodpovědné za správné přiřazení SELinux uživatele na klasického linuxového uživatele. Pokud se při přihlášení narazí na SELinux uživatele se stejným identifikátorem jako má linuxový, je tento přiřazen výchozímu terminálovému procesu. Definovat každého uživatele zvlášť v bezpečnostních pravidlech by bylo velmi zdlouhavé a u systémů s velkým počtem účtů i náročné. Běžní uživatelé proto mají stejná práva s ohledem na SELinux – roli `user_r` a výchozí doménový typ `user_t`. SELinux má speciálního uživatele označovaného `user_u`. Pokud se definice tohoto uživatele objeví v pravidlech, jsou všichni linuxoví k němu přiřazeni, ale pouze v případě pokud nemají žádného odpovídajícího SELinuxového uživatele definovaného v pravidlech.

Tudíž pokud se v *policy.conf* objeví řádek vypadající následovně

```
user user_u roles { user_r };
```

je definován obecný uživatel, kterému je přiřazena role `user_r` – stejná jako výše v definici pro uživatele *george*. Účet *george* již je v pravidlech nadefinován, proto se na něj definice obecného uživatele `user_u` nevztahuje. Ale pokud existuje například ještě uživateslký účet *michael*, bude mu přiřazen všeobecný uživatel, protože nemá vlastní definici.

Pokud se uživatel snaží přihlásit a není definováno ani pravidlo pro jeho účet, ani všeobecný úživatel `user_u`, je vyzván k zadání platného bezpečnostního kontextu, který musí obsahovat i identifikaci uživatele. Pokud není v pravidlech definován žádný SELinux uživatel ani `user_u`, není možné se vůbec na tento systém přihlásit, protože není žádný identifikátor uživatele, který by bylo možno použít. Pro tuto příležitost je dobré si v pravidlech nadefinovat obecného uživatele. Není potom nutné definovat pro každého uživatele zvláštní nastavení.

SELinux má také druhého speciálního uživatele `system_u`, který se využívá pro všechny systémové procesy, jimž je například `init` a démoni, kteří jsou jím spouštěni. Není úplně nutné tohoto uživatele definovat, ale je velmi doporučováno ho do pravidel umístit [3]. Zároveň je nutné, aby nebyl vytvořen žádný linuxový uživatel se jménem `system_u`, protože ten by měl po přihlášení poměrně velké oprávnění.

4.3.4 Omezení

SELinux obsahuje mechanizmus, který umocňuje omezení přístupu povoleného bezpečnostními pravidly bez ohledu na povolovací pravidla `allow`.

Definice omezení

Pravidlo `constraint` má tři základní prvky: množinu tříd objektů, na které dané omezení platí; skupiny oprávnění pro třídy, na kterých bude vynuceno; logický výraz tohoto omezení.

Pravidlo začínající klíčovým slovem `constraint` umožňuje omezit specifikovaná oprávnění pro uvedené třídy objektů. A to definicí omezení založených na vztahu mezi zdrojovým a cílovým bezpečnostním kontextem. Teď si uvedeme plnou verzi zápisu tohoto pravidla:

`constraint` *`množina_tříd skupina_oprávnění výraz`*

- **množina_tříd** – jedna nebo více tříd objektů; zápis více položek se uvádí do složených závorek a jednotlivé prvky jsou oddělené mezerou; nejsou povoleny speciální operátory jako $*$, $-$ a $\sim$
- **skupina_oprávnění** – uvedeno jedno nebo více povolení, přičemž všechna musí být platná pro všechny třídy objektů v tomto zápise uvedených; vícenásobný zápis již standardně psaný do složených závorek a prvky oddělené mezerami; zde také nejsou povoleny speciální operátory $*$, $-$ a $\sim$
- **výraz** – logický výraz pro toto omezení

V definici logického výrazu se používají tyto prvky:

t1, r1, u1	zdrojový typ, role nebo uživatel
t2, r2, u2	cílový typ, role či uživatel

V logických výrazech se dále užívají tyto operátory:

==	určení členu nějaké skupiny nebo také rovnost
!=	není členem nebo nerovnost
eq	rovnost (pouze u rolí)
dom	nadřazenost (pouze u rolí)
domby	podřazený nějaké roli (jen u rolí)

incomp neporovnatelné (jen u rolí)

Výrazy definující omezení mohou být velmi komplexní, protože lze definovat jakoukoliv kombinaci mezi třemi základními prvky (uživatelem, rolí a typem). Tyto výrazy porovnávají kontext zdrojového procesu a cílového objektu mezi sebou a nebo s určitými pojmenováními (jako identifikátor role či typu).

```
constraint process transition (u1 == u2);
```

U tohoto omezení se staráme pouze o třídy objektu *process* a o povolení přechodu mezi doménami. Z logického výrazu (u1==u2) vyplývá, že je potřeba, aby zdrojová a cílová identifikace uživatele byla shodná. Potom teprve může dojít ke přechodu z jedné domény do jiné.

Složitěji vypadá tento výraz:

```
constrain process transition (u1 == u2 and r1 == r2);
```

Princip je však poměrně jasný – pro změnu mezi doménami u tohoto procesu musí být splněny podmínky, že zdrojový i cílový uživatel mají stejnou identifikaci a zároveň i zdrojové a cílové označení role je shodné.

Toto byl pouze lehký úvod do definice omezení, více je možné najít například v [2] nebo [4]. Blíže se omezením nebudeme věnovat, protože běžný správce SELinuxu se k definicím této části nedostane – jedná se spíše o volitelnou vlastnost.

4.3.5 Značení objektů

Značení je velmi důležitou částí SELinuxu. Je prováděno kvůli tomu, aby byla nadefinovaná bezpečnostní politika funkční. Každá instance objektu musí být označena příslušným bezpečnostním kontextem.

Ukázkou dobře označeného souboru může být výpis souboru `/etc/shadow`:

```
# ls -Z /etc/shadow
-r-------  root   root  system_u:object_r:shadow_t  shadow
```

Tomuto souboru je přiřazen bezpečnostní kontext `system_u:object_r:shadow_t`. Bezpečnostní kontext je jediným atributem, který SELinux využívá v rozhodování o povolení přístupu.

Definice značení jednotlivých souborů se nachází v adresáři `file_context` a například pro démona ntpd je definice následující:

```
/var/lib/ntp(/.*)?     system_u:object_r:var_lib_ntp_t
/etc/ntp.conf          system_u:object_r:etc_ntp_t
/usr/sbin/ntpd         system_u:object_r:ntpd_exec_t
```

První sloupec obsahuje regulární výraz, který specifikuje soubory na disku. Druhý sloupec říká, jaký kontext bude daným souborům přiřazen.

Po definici značení souborů je třeba celý systém souborů přeznačit a to se provede příkazem `make relabel` nebo také pomocí `fixfiles relabel`.

4.4 Monitorování v SELinuxu

Velmi důležitou věcí je způsob zjišťování, že k nějakému povolení či zamezení přístupu došlo. SELinux ukládá každou událost, která je označena pro zaznamenání do systémového logu. Tyto informace se označují jako AVC zprávy.

4.4.1 Formát AVC zpráv

Každá taková zpráva vznikne pokud nastane v činnosti SELinuxu nějaká událost, která má nastaveno, že se má zaznamenat. Základním formátem takové AVC je tento:

```
avc: result { operation } for pid=PID exe=EXE path=PATH
     dev=DEVNO:PTO ino=NODE scontext=SOURCE tcontext=TARGET
     tclass=CLASS
```

Význam jednotlivých částí této zprávy:

- **result** – hodnota *granted* pro povolení přístupu nebo *denied* pro zákaz
- **operation** – operace, při které k události došlo. SELinux definuje asi 150 takových operací. Příkladem může být *read* pro čtení či *write* pro zápis. Všechny operace lze nalézt například v příloze B literatury [2].
- **PID** – ID procesu, který způsobil tuto akci
- **EXE** – absolutní cesta ke spouštěcímu souboru, který se o událost pokusil
- **PATH** – absolutní cesta objektu, na který byla akce zkoušena
- **DEVNO:PTNO** – číslo blokového zařízení, na kterém byla operace prováděna a přesné číslo oddílu
- **NODE** – číslo označené jako inode (každý objekt má jedinečné číslo) objektu, ke kterému bylo přistupováno
- **SOURCE** – bezpečnostní kontext procesu, který zápis do logu způsobil
- **TARGET** – kontext cílového objektu

- **CLASS** – třída cílového objektu, SELinux obsahuje cca 30 těchto tříd a lze je nalézt například v příloze A literatury [2] (příkladem může být typ *file* označující soubor či *dir* značící adresář)

Nejlepší na učení je praktický rozbor takové zprávy, proto se teď pusťme na tuto AVC zprávu:

```
avc: denied { write } for pid=1400 exe=/usr/bin/nmap lport=255
     scontext=root:staff_r:nmap_t tcontext=root:staff_r:nmap_socket_t
     tclass=rawip_socket
```

Zde je ukázáno zamezení zápisu. Proces snažící se o zápis je `/usr/bin/nmap` a má číslo ID 1400. Bezpečnostní kontext tohoto procesu je *root:staff_r:nmap_t* a kontext cílového objektu je *root:staff_r:nmap_socket_t*. Objekt, na kterém je pokus o provedení, má třídu *rawip_socket*, která značí normální IP soket. Dále je zde vidět ještě logický zdrojový port 255. Výsledkem je zpráva, že bezpečnostní systém zabránil aplikaci Nmap zápis na socket.

4.4.2 Umístění záznamů

V základní instalaci operačního systému, který nepochází od společnosti RedHat, se systémové záznamy ukládají do standardního souboru se zprávami o činnosti systému – a tímto je `/var/log/messages`, který je přístupný pomocí zadání příkazu `dmesg`. Pokud tedy chceme na obrazovku získat výpis pouze AVC zpráv v systémových záznamech, použijeme příkaz `dmesg | grep avc`.

Toto standardní umístění není příliš výhodné, protože formáty se v každé distribuci liší. Proto je dodáván programový balíček `auditd`, který sjednocuje formáty AVC zpráv a umísťuje je do samostatného souboru mimo další systémové záznamy – tímto souborem poté je `/var/log/audit/audit.log`. Tento balíček je automaticky nainstalován v distribucích společnosti RedHat (RHEL, CentOS, Fedora), v ostatních distribucích je většinou potřeba jej ručně doinstalovat.

A jak vlastně vypadá celkový formát AVC zprávy vytvořené pomocí balíku auditd? Ukažme si to na následujícím výpisu jedné z mnoha zpráv:

```
type=AVC msg=audit(1206473715.033:1644): avc:  denied  { search }
   for  pid=9657 comm="dpkg" name="files" dev=sda3 ino=55474
   scontext=system_u:system_r:dpkg_t
   tcontext=system_u:object_r:file_context_t
   tclass=dir
```

Význam jednotlivých polí by měl být jasný vzhledem k dřívějšímu vysvětlení obecného formátu zprávy. Za zmínku však stojí, že položky *name, dev, ino* jsou nepovinné a nemusí se objevit u každého záznamu.

4.5 Psaní pravidel

Proto, abychom mohli začít psát vlastní přístupová pravidla, je potřeba nainstalovat ještě zdrojové soubory s pravidly SELinuxu – v jednotlivých systémech se název správného balíčku může lišit, ale vždy to bude něco jako `selinux-devel`, `selinux-policy-devel` či `selinux-source` s možnými obměnami (tzn. že může například obsahovat název druhu pravidel).

Při psaní pravidel máme dvě možnosti, jak je vytvářet (i když brzy zjistíme, že možnosti jsou v podstatě tři):

- ruční psaní nových pravidel
- využití utility nazvané `audit2allow`

Každá možnost psaní pravidel má své pro i proti. Proto si zkusme nyní přiblížit jednotlivé volby.

4.5.1 Ruční psaní pravidel

Jedná se o poměrně náročnou činnost. Zvláště pro uživatele, který ještě není příliš zběhlý v systému SELinux a psaní jeho pravidel. A i když do SELinux pronikne, tak stále může zjišťovat, že ruční psaní není vůbec jednoduchá věc a může zabrat spoustu času. Na druhou stranu je poté získán systém, který má bezpečnost „ušitu“ na míru danému programovému vybavení a útok na takto zabezpečený počítač je v podstatě téměř nemožný.

Nyní si v několika krocích ukažme, jak vypadá základní konfigurace. Vše bude probíhat v adresáři se zdrojovými soubory pravidel. Ty jsou umístěny v každém systému jinde. Budeme se zajímat pouze o systém z rodiny RedHat (CentOS) a systém z rodiny Debian (Ubuntu):

- CentOS: `/usr/share/selinux/devel/`
- Ubuntu: `/usr/share/selinux/policy/current/`

V tomto adresáři a podadresářích jsou uloženy veškeré soubory, které jsou potřeba pro kompilaci vlastních pravidel. Systém CentOS zde také obsahuje soubor `example.te`, který slouží jako základní ukázka pro psaní nového pravidla.

Pro tvorbu nových pravidel bude nejlepší vytvořit nový podadresář nazvaný například `local` a do něj teprve vytvářet nové podadresáře, které je dobré pojmenovávat podle programu či účelu, k němuž slouží. V tomto adresáři je také umístěn samotný soubor Makefile, který nám nabízí možnosti kompilace, které byly popsány v kapitole 4.3, ale pro osvěžení si popišme ještě ty nejdůležitější:

- **load** – zkompiluje, vytvoří, a nahraje novou politiku do jádra
- **reload** – zkompiluje, vytvoří novou politiku a pokud byla změněna, tak ji nahraje do jádra
- **clean** – smaže pomocné moduly a dočasný adresář
- **bez parametru** – zkompiluje zdrojové soubory v aktuálním adresáři

Při tvorbě pravidel se používají celkem tři druhy souborů s příponami *.fc*, *.if* a *.te*. Nyní si postupně probereme všechny typy a popíšeme, co a jak se do nich píše.

Soubory .fc V nich se specifikuje, jakým bezpečnostním kontextem mají být soubory označeny. Základní formát je následující:

```
cesta [prepinac] gen_context(novy_kontext, level)
```

Cesta bývá udávána jako absolutní nebo formou regulárního výrazu – odkazujeme se tak buď na soubor či adresář. Pokud je přepínač vynechán, budou označkovány všechny soubory a adresáře. Při použití přepínače `--` se označkují pouze soubory a při využití přepínače `-d` jen adresáře. Stávající soubory si ponechají svůj kontext a nově vytvořené jej zdědí od nadřazeného adresáře.

Soubory .if Do těchto souborů se definují vlastní makra, která jsou napsána pomocí makro procesoru m4. Tato makra poté slouží ke zpřehlednění pravidel. Definice maker není povinná a závisí pouze na administrátorovi, zda tento typ souborů použije.

Nyní si ukažme jednoduchou definici makra:

```
interface(‘spousteni’,‘
   gen_require(‘
      # typy využívané v˜makru
      type bin_t;
   ’)

   # přístupová pravidla
   allow $1 bin_t: file { read getattr execute };
’)
```

Toto jednoduché makro se volá například pomocí *spousteni(user_t)*. Parametr $1 je poté nahrazen typem user_t.

Soubory .te Tento druh souborů je pro celou konfiguraci vůbec nejdůležitější, protože se do něj definují samotná pravidla TE. Zde je možné při definici využít také přednastavená či námi vytvořená makra umístěná v souborech s příponou .if.

Ukázkový soubor by mohl vypadat například následovně:

```
policy_module(jmeno_modulu, verze_modulu);

require{
   # zde se nachází definice tříd objektů, atributů
   # a typů, které už jsou v~systému známy

   atrribute domain;
   type bin_t, user_t;
   class file { read getattr execute };
};

# deklarace nových typů
type new_type_t, domain;

# nová pravidla
# povolit doméně s~typem new_type_t spustit
# nebo číst soubory s~typem bin_t
allow new_type_t bin_t: file { read getattr execute };

# použité makra(parametry)
# po spuštění souboru s~typem bin_t povol přechod
# z~domény user_t do domény new_type_t
domain_auto_trans(user_t, bin_t, new_type_t)
```

Nejdříve se v hlavičce definuje samotný název a verze námi vytvořeného modulu. Poté se do direktivy *require* uvedou typy, třídy objektů a atributy, které jsou již v systému definovány a budeme je při psaní našeho modulu využívat. Poté se již mohou definovat naše nové typy, třídy objektů a pravidla. Lze samozřejmě používat libovolně systémová či námi vytvořená makra.

Kompilace a zavedení nových pravidel do jádra

Po vytvoření souborů s novými pravidly je ještě potřeba jejich kompilace a zavedení do jádra systému SELinux. Postup při této činnosti je následující:

- Přeložení pravidel pomocí souboru Makefile uloženého v adresáři se zdrojovými soubory pravidel SELinuxu. Volání se provádí z adresáře, kam jsme psali nová

pravidla pomocí následujícího příkazu (cestu k souboru Makefile je případně potřeba upravit tak, aby odpovídala používanému operačnímu systému; tato ukázka počítá se systémem CentOS):

```
make -f /usr/share/selinux/devel/Makefile
```

- Tímto se pravidla pouze zkompilují do souboru s příponou .pp, ale nezavedou se do jádra. Ruční zavedení, ale i odstranění či výpis modulů v jádře se prování pomocí příkazu `semodule`.
  - zavedení modulu do jádra: `semodule -i jmeno_modulu.pp`
  - odstranění modulu z jádra: `semodule -r jmeno_modulu`
  - výpis modulů aktuálně zavedených v jádře: `semodule -l`
- Po zavedení modulu by se nové kontexty se souboru .fc měly objevit v souboru `file_contexts`, který se umístěn v adresáři se zavedeným druhem pravidel – v CentOS `/etc/selinux/targeted/contexts/files/file_contexts`.

Přeznačení souborového systému

Aby měly všechny soubory v systému aktuální kontext s ohledem na námi vytvořené změny, je potřeba souborový systém přeznačit. K této činnosti slouží příkaz `fixfiles` s parametrem `relabel`:

- přeznačení celého systému souborů: `fixfiles relabel`
- přeznačení vybraného adresáře: `fixfiles relabel /home/`

Pro přeznačení jediného souboru se však používá jiný příkaz a to `restorecon /cesta/k/souboru/nazev_souboru`.

4.5.2 Utilita Audit2allow

V začátcích konfigurace můžeme dostávat mnoho zpráv o zamezení přístupu. Kontrolovat každou zvlášť a poté podle ní zapisovat pravidlo pro povolení by bylo velmi zdlouhavé. Proto je nabízena možnost využít nástroje `audit2allow`, který prohlédne záznamy a z operací *denied* vygeneruje povolení pomocí pravidla *allow*. Zde se tedy dostáváme k druhé a i třetí možnosti psaní pravidel – je možné vygenerovat pravidla plně automaticky bez zásahu uživatelem nebo si vygenerovat soubor .te a ten upravit.

Plně automatická tvorba

Tato možnost se může zdát pro začínajícího uživatele SELinuxu jako nejlepší volba. Opak je však pravdou, protože při této činnosti nemá vůbec žádný přehled o tom, co se vlastně povoluje. Také se velmi často stává, že kompilace takového modulu skončí chybou a poté je stejně odkázán na úpravu vygenerovaných pravidel.

Nyní si objasněme, jak použít plně automatickou tvorbu pravidel:

- Automatické vytvoření nového modulu s názvem *modul*:

  ```
  cat /var/log/audit/audit.log | audit2allow -M modul
  ```

- Po spuštění příkazu by se měl objevit výpis podobný následujícímu, kdy je nejdříve oznámeno, že se vytvořil soubor .te, poté zkompilován pomocí programu checkmodule a následně vytvořen balíček modulu:

  ```
  Generating type enforcment file: modul.te
  Compiling policy: checkmodule -M -m -o modul.mod modul.te
  Building package: semodule_package -o modul.pp -m modul.mod
  ```

- Pokud vše proběhne v pořádku, je potřeba ještě modul nahrát do jádra (toto automaticky neprobíhá) za pomoci již zmíněného příkazu:

  ```
  semodule -i modul.pp
  ```

Poloautomatická tvorba pravidel

Vytváření nových pravidel touto cestou je asi nejlepší kompromis mezi složitostí psaní a bezpečností systému. Soubor .te je generován za pomoci nástroje `audit2allow`, ale neprobíhá již dále kompilace modulu, protože se počítá s tím, že si správce systému tento soubor prohlédne, udělá v něm případné změny a až poté jej zkompiluje a nahraje do jádra systému SELinux.

Tento postup lze popsat v těchto několika krocích:

- Vytvoření nového souboru .te s název *modul*:

  ```
  cat /var/log/audit/audit.log | audit2allow -m modul > modul.te
  ```

- Otevření a případná úprava nově vzniklého souboru:

  ```
  nano modul.te
  ```

- Zkompilování nového modulu:

  ```
  checkmodule -M -m -o modul.mod modul.te
  ```

- Vytvoření balíčku modulu:

  ```
  semodule_package -o modul.pp -m modul.mod
  ```

- Nahrání nově vytvořeného modulu do jádra SELinuxu:

  ```
  semodule -i modul.pp
  ```

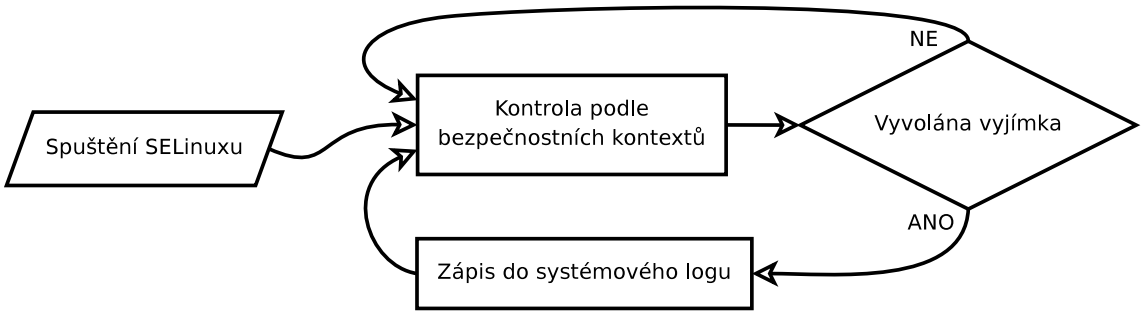
Občas se může stát, že některá část překladu skončí chybou – většinou to bývá chybným zápisem ve vytvořeném .te souboru. V tomto případě je potřeba soubor znovu otevřít a upravit příslušný chybný řádek. Z výpisu chyby při kompilaci či vytváření balíčku se dá většinou poměrně snadno určit, kde je chyba.

5 PROGRAM NA MONITOROVÁNÍ A PSANÍ PRAVIDEL

Praktická část práce je věnována vývoji aplikace, která zpracuje záznamový soubor systému a analyzuje v něm nalezené AVC zprávy. Pomocí grafického rozhraní by bylo uživateli umožněno dekódování jednotlivých zpráv a bylo by na jeho zvážení, zda takovou zprávu ignorovat nebo se pokusit podle ní zadat nové pravidlo či využít utility `audit2allow`, která by zajistila automatické zpracování zamítnutých pokusů o přístup a pomocí pravidla *allow* by je povolila. Dále by bylo uživateli umožněno vytváření úplně nových pravidel, protože je možné, že bude také provozována aplikace, která není v základní sadě pravidel definována.

5.1 Základní návrh činnosti

Nejlépe je si zamýšlenou činnost tohoto programu ukázat na několika blokových schématech. Na obr. 5.1 je vidět, jak zhruba vypadá vygenerování události do systémového logu.

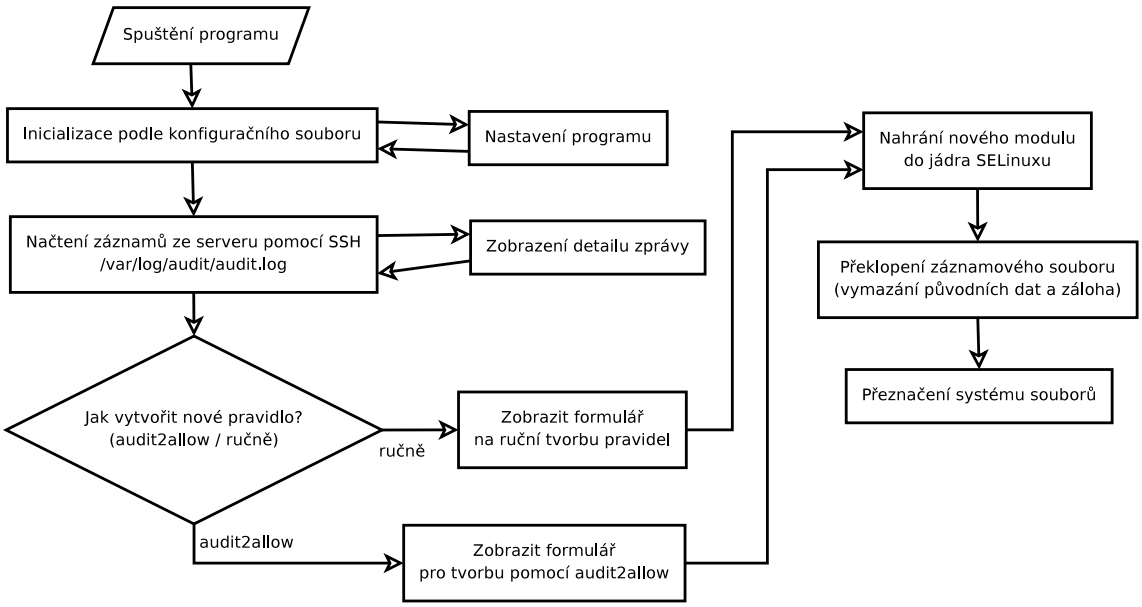


Obr. 5.1: Nástin činnosti SELinuxu

Na obrázku 5.2 je vidět, jak zhruba vypadá schéma činností programu.

5.2 Využití programu

Tato aplikace najde své využití všude, kde je potřebná úprava pravidel či prohlížení záznamů systému SELinux. Výhodná je hlavně pro správce operačních systémů Linux. Aplikace má jednoduché grafické rozhraní a veškerá činnost probíhá vzdáleně na síťovém serveru pomocí protokolu SSH. Při výuce na Ústavu telekomunikací může tento program najít praktické nasazení v předmětu „Návrh, správa a bezpečnost počítačových sítí“, kde by posluchači mohli získat základní znalosti o řízení



Obr. 5.2: Vývojový diagram programu

přístupu k souborům a procesům v operačním systému Linux. Při výuce se v době psaní práce využívá virtuální počítač, který simuluje reálný síťový server.

5.3 Způsob implementace

Základním problémem je výběr programovacího jazyka. Jelikož je vyžadováno, aby byl systém vytvořen v grafickém prostředí, a to pomocí jazyka, který má poměrně jednoduchou a srozumitelnou syntaxi (s ohledem na případné úpravy a rozšiřování funkčnosti), byl pro realizaci této aplikace zvolen programovací jazyk Python. Jedná se o objektově orientovaný programovací jazyk, který je dnes obsažen v téměř každé instalaci systému Linux a je dostupný také pro OS rodiny Windows.

Při výběru grafické knihovny je několik možností volby (Tk/Tcl, QT, GTK). Každá má své výhody i nevýhody. U Tk/Tcl je to například rozšiřitelnost a přenositelnost, ale naopak poměrně zastaralý vzhled a složité programování některých konstrukcí. U knihoven QT a GTK je sice složitější implementace, ale mají modernější vzhled a daleko vyšší uživatelskou přívětivost. Tato aplikace bude proto psána za pomoci knihovny GTK.

Na operační systém je kladeno několik požadavků – jedním ze základních a velmi důležitých je výběr linuxové distribuce. Dále je potřeba doinstalovat příslušné knihovny umožňující vývoj nového programu a zvolit vývojové prostředí, ve kterém bude program vznikat.

5.3.1 Výběr Linuxové distribuce

Dnes již není takový problém najít distribuci, ve které by byla technologie SELinux obsažena, ale najdou se i systémy, do kterých je instalace velmi složitá a někdy i téměř nemožná.

Nejjednodušší je samozřejmě použití distribuce přímo od společnosti RedHat – buď komerční RedHat Enterprise Linux nebo jednu z jejich dvou volně dostupných větví, které využívají balíčky z této enterprise distribuce, a to Fedora či CentOS. Aplikace bude testována na systému CentOS a také systému Debian, jelikož se nyní právě tyto dva využívají k výuce předmětu MNSB.

5.3.2 Příprava systému

Do systému je třeba nainstalovat příslušné knihovny, které umožní jak změnu pravidel SELinuxu, tak hlavně naprogramování aplikace v jazyce Python se vzhledem pomocí knihoven GTK. V Pythonu se používá pro tuto grafiku knihovna PyGTK a doinstalování její podpory není složité – slouží k tomu balíček `python-pygtk`. Dále je potřeba instalace zdrojových kódů technologie SELinux, což se provede nainstalováním balíku `selinux-policy-devel`.

Dále je vhodné, aby AVC zprávy byly odděleny od systémových záznamů a to nainstalováním programového balíku `auditd` – aplikace s takto umístěnými záznamy počítá.

5.4 Vzdálené připojení pomocí SSH

Jelikož veškerá činnost programu bude probíhat na serveru vzdáleně pomocí protokolu SSH, bude užitečné ukázat základy práce s tímto protokolem.

Základní použití pro připojení na vzdálený server se provádí příkazem `ssh`:

```
ssh root@192.168.2.2
```

Zde se na vzdálený server s IP adresou 192.168.2.2 snažíme připojit pomocí uživatelského jména *root* (jako správce vzdáleného systému). S ohledem na zvolený způsob autentizace jsme buď dotázáni na heslo nebo ověření proběhne pomocí klíčů. Použití klíčů je samozřejmě velmi doporučováno, protože je mnohem bezpečnější a na druhou stranu pro uživatele i příjemnější, neboť nemusí neustále zadávat své heslo.

Jak vytvořit nové klíče? Tato činnost není nijak složitá, poslouží nám k tomu následující příkaz:

```
ssh-keygen [volby]
```

Nejdůležitější volbou bývá `-t` pro určení typu klíče *rsa* či *dsa* (dnes je doporučování typ dsa) a dále volba `-b` pro bitovou délku klíče (většinou není potřeba zadávat a měnit základní délku). Po spuštění tohoto příkazu je uživatel dotázán na několik otázek – mezi ně patří i heslo ke klíči. Zde je volba již na uživateli – normálně se heslo ke klíči neuvádí, protože poté odpadá výhoda, že není potřeba zadávat při každém připojení heslo. Na druhou stranu pokud není privátní klíč zaheslován a dostane se k němu třetí osoba, může jej jednoduše zneužít. Spíše se heslo neuvádí a uživatel se snaží maximálně si privátní klíč hlídat, aby se k němu nedostala nežádoucí osoba.

Nové klíče jsou vytvořeny v domovském adresáři v podadresáři `.ssh` a mají název `id_dsa` pro privátní klíč a `id_dsa.pub` pro veřejný klíč (při standardním umístění a volbě typu dsa). Privátní klíč musí být dobře střežen a hlídán před zneužitím. Otisk veřejného klíče z `id_dsa.pub` se umístí na server, kam se chceme pomocí našeho klíče připojovat. Zapíše se do souboru `authorized_keys`, který je umístěn v adresáři `.ssh` vzdáleného uživatele.

Poté již můžeme vyzkoušet připojení pomocí prvního příkazu a neměli bychom již být dotázáni na heslo, ale přímo spojeni. Je možné také pomocí přepínače `-i` určit, který privátní klíč se má použít:

```
ssh -i /cesta/ke/klici/id_dsa_moje root@192.168.2.2
```

Této možnosti bude využíváno v aplikaci, protože se předpokládá, že na pracovní stanici může být například při výuce několik privátních klíčů různých uživatelů.

Je vhodné si ukázat, že pomocí jediného SSH příkazu jde pomocí tzv. roury spouštět příkaz či skupinu příkazů stejně jako bychom byli přímo připojeni ke vzdálenému stroji. Pomocí následujícího příkazu se pouze připojíme na nezbytně nutnou dobu a vykonáme akci uvedenou na konci příkazu (v tomto případě jde o výpis domovského adresáře uživatele root):

```
ssh -i /cesta/ke/klici/id_dsa_moje root@192.168.2.2 ls
```

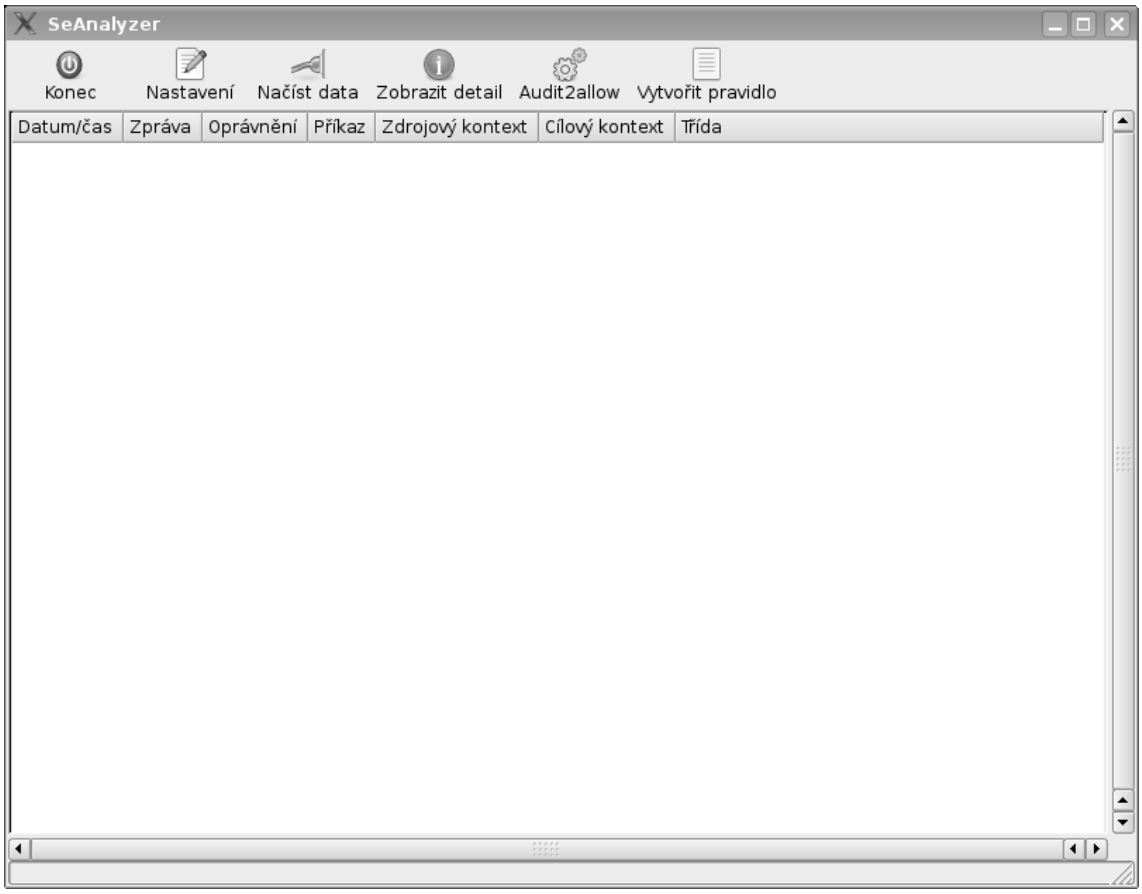
Výstupem bude seznam souborů a adresářů v požadovaném adresáři.

5.5 Grafické rozhraní programu

Grafické rozhraní programu je velmi jednoduché, aby nebyl problém jej používat i pro začínajícího uživatele. V podstatě se bude skládat ze 3 základních částí:

1. část s tlačítkovou lištou
2. hlavní část okna programu zobrazující analyzované AVC zprávy
3. stavová lišta informující o aktuálním stavu a doplňujících informacích

Toto základní rozvržení je možné vidět na obr. 5.3, kde je ukázáno hlavní okno programu.



Obr. 5.3: Hlavní okno programu

5.6 Základní práce s programem

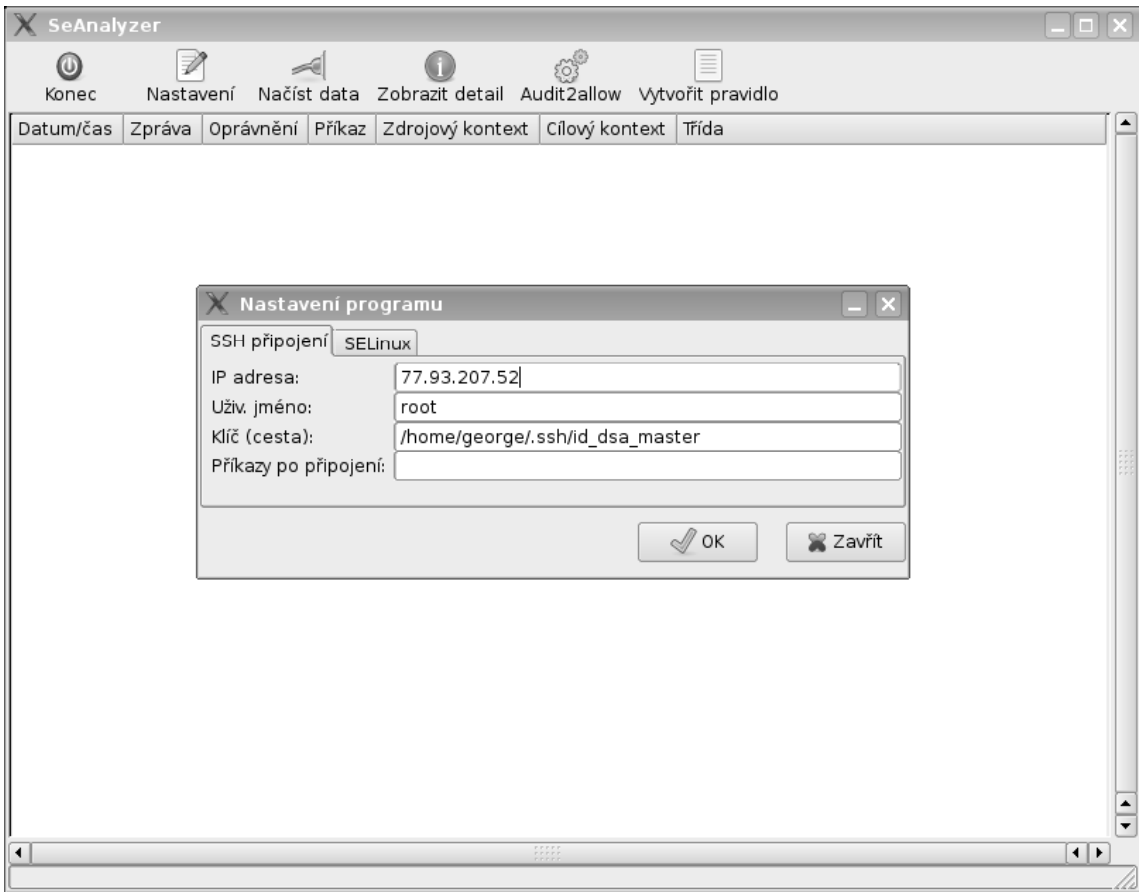
V dalších odstavcích si popíšeme, jak se s programem dá pracovat po uživatelské stránce. Funkční stránka, včetně základního principu „pozadí“ aplikace, bude probrána v další části.

Po spuštění samotného programu se uživateli zobrazí pouze prázdné okno a následuje několik činností, které musí uživatel udělat.

5.6.1 Nastavení programu

Klepnutím na tlačítko „Nastavení“ se vyvolá nabídka zobrazená na obr. 5.4. Pokud již uživatel dříve nastavení provedl, je uloženo v souboru `seanalyzer.conf` v pracovním adresáři, odtud je načteno do formuláře. V opačném případě se uživateli

zobrazí prázdný formulář.

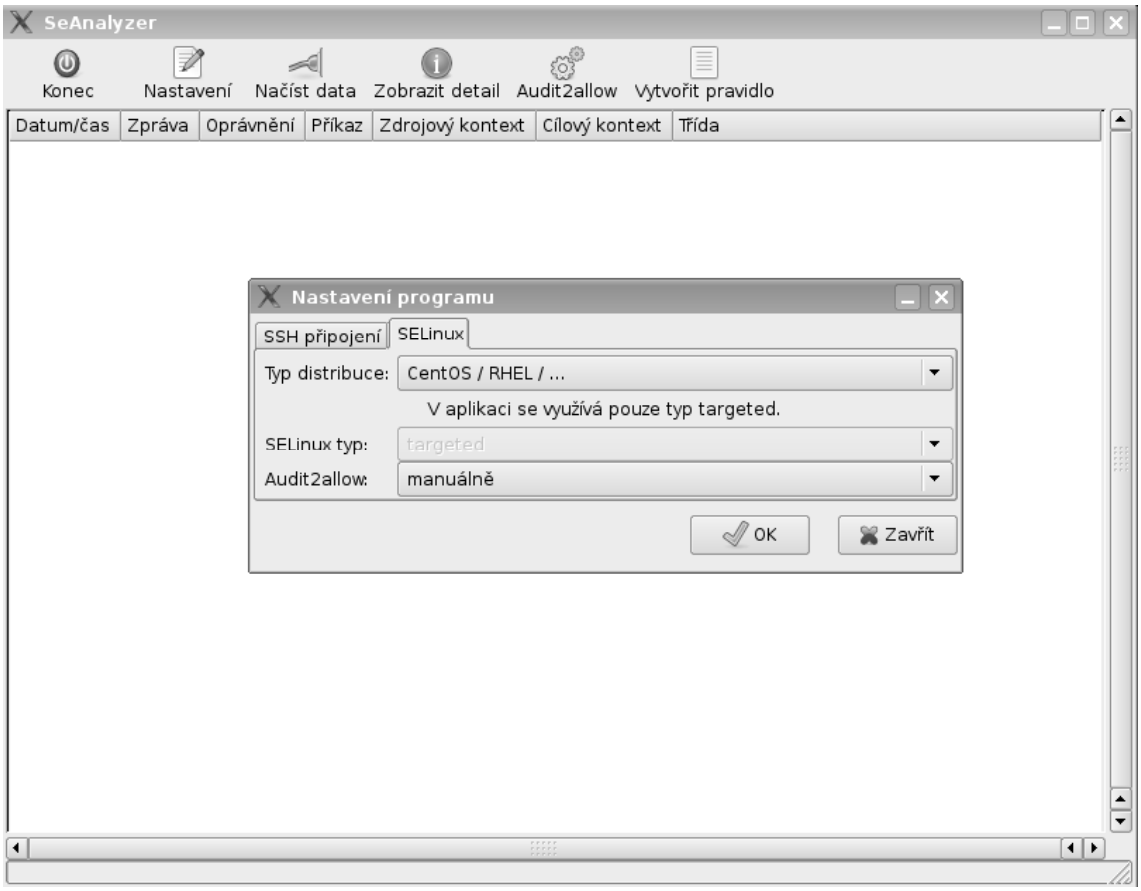


Obr. 5.4: Nastavení programu – SSH připojení

Samotné nastavení se skládá ze dvou záložek – v první se nastavuje vše potřebné k SSH připojení a v druhé informace týkající se distribuce Linuxu a pár informací o systému SELinux běžícím na serveru.

Záložka na nastavení parametrů SSH je vidět na obr. 5.4 a vyplňují se tyto položky:

- IP adresa serveru, který chceme analyzovat a vytvářet na něm nová pravidla
- uživatelské jméno na vzdáleném počítači (nejčastěji to bude právě správce systému *root*)
- absolutní cesta k privátnímu klíči, který chceme použít pro připojení
- dodatečné příkazy nejsou pro činnost programu nutné – mohou se použít například, pokud chceme nejdříve zálohovat některé soubory, s nimiž budeme pracovat



Obr. 5.5: Nastavení programu – distribuce, SELinux

V druhé záložce se nastavují základní parametry systému SELinux a použité distribuce (možno vidět na obr. 5.5):

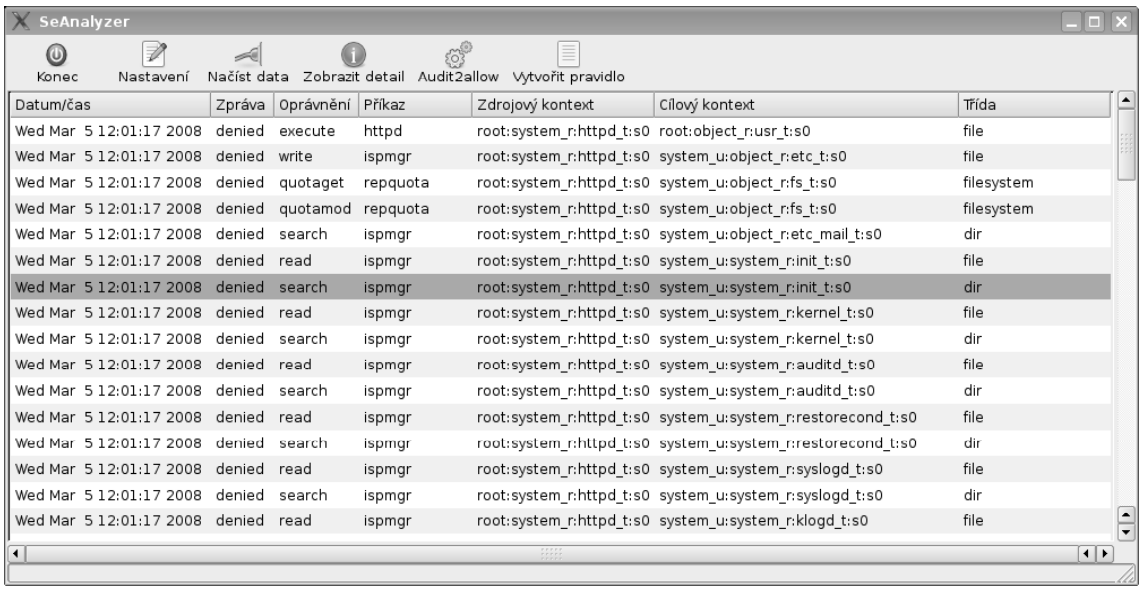
- typ použité distribuce – podle něj se prohledává správný adresář se zdrojovou politikou SELinuxu
- jak se budou při použití audit2allow generovat nová pravidla

Po provedení nastavení a potvrzení formuláře je do souboru `seanalyzer.conf` zvolená konfigurace uložena a při činnosti programu z tohoto souboru automaticky načítána.

5.6.2 Načtení záznamů ze serveru

Po klepnutí v tlačítkové liště na tlačítko „Načíst data“ se program pokusí připojit na zvolený server pomocí nastaveného uživatelského jména a klíče, načíst a zpracovat soubor `/var/log/audit/audit.log`. Do hlavní části programového okna jsou postupně vkládány jednotlivé záznamy. Sloupce tabulky mají následující význam:

- **Datum/čas** – datum a čas, kdy byl záznam vytvořen
- **Zpráva** – buď informace o povolení (*allow*) nebo zamezení přístupu (*denied*). Jelikož se většinou zaznamenávají pouze zamezení přístupu, převážná část řádků bude obsahovat zprávu *denied*.
- **Oprávnění** – činnost, kterou se daný program pokusil vykonat a při které výjimka vznikla
- **Příkaz** – systémový příkaz (program), který daný záznam způsobil
- **Zdrojový kontext** – zdrojový kontext objektu
- **Cílový kontext** – kontext, na který se snažil objekt přejít
- **Třída** – třída objektu, na který se přistupovalo



Datum/čas	Zpráva	Oprávnění	Příkaz	Zdrojový kontext	Cílový kontext	Třída
Wed Mar 5 12:01:17 2008	denied	execute	httpd	root:system_r:httpd_t:s0	root:object_r:usr_t:s0	file
Wed Mar 5 12:01:17 2008	denied	write	ispmgr	root:system_r:httpd_t:s0	system_u:object_r:etc_t:s0	file
Wed Mar 5 12:01:17 2008	denied	quotaget	repquota	root:system_r:httpd_t:s0	system_u:object_r:fs_t:s0	filesystem
Wed Mar 5 12:01:17 2008	denied	quotamod	repquota	root:system_r:httpd_t:s0	system_u:object_r:fs_t:s0	filesystem
Wed Mar 5 12:01:17 2008	denied	search	ispmgr	root:system_r:httpd_t:s0	system_u:object_r:etc_mail_t:s0	dir
Wed Mar 5 12:01:17 2008	denied	read	ispmgr	root:system_r:httpd_t:s0	system_u:system_r:init_t:s0	file
Wed Mar 5 12:01:17 2008	denied	search	ispmgr	root:system_r:httpd_t:s0	system_u:system_r:init_t:s0	dir
Wed Mar 5 12:01:17 2008	denied	read	ispmgr	root:system_r:httpd_t:s0	system_u:system_r:kernel_t:s0	file
Wed Mar 5 12:01:17 2008	denied	search	ispmgr	root:system_r:httpd_t:s0	system_u:system_r:kernel_t:s0	dir
Wed Mar 5 12:01:17 2008	denied	read	ispmgr	root:system_r:httpd_t:s0	system_u:system_r:auditd_t:s0	file
Wed Mar 5 12:01:17 2008	denied	search	ispmgr	root:system_r:httpd_t:s0	system_u:system_r:auditd_t:s0	dir
Wed Mar 5 12:01:17 2008	denied	read	ispmgr	root:system_r:httpd_t:s0	system_u:system_r:restorecond_t:s0	file
Wed Mar 5 12:01:17 2008	denied	search	ispmgr	root:system_r:httpd_t:s0	system_u:system_r:restorecond_t:s0	dir
Wed Mar 5 12:01:17 2008	denied	read	ispmgr	root:system_r:httpd_t:s0	system_u:system_r:syslogd_t:s0	file
Wed Mar 5 12:01:17 2008	denied	search	ispmgr	root:system_r:httpd_t:s0	system_u:system_r:syslogd_t:s0	dir
Wed Mar 5 12:01:17 2008	denied	read	ispmgr	root:system_r:httpd_t:s0	system_u:system_r:klogd_t:s0	file

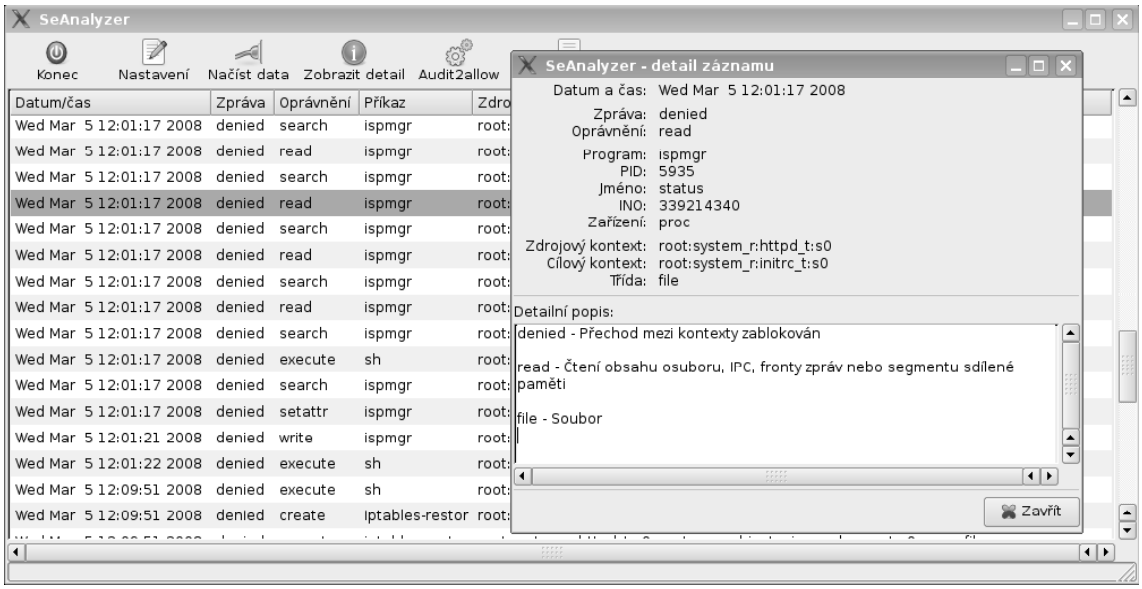
Obr. 5.6: Načtené záznamy ze serveru

Na obr. 5.6 je vidět již načtený a zpracovaný soubor. V jednotlivých řádcích jsou postupně všechny nalezené záznamy.

5.6.3 Zobrazení detailu záznamu

Přestože mohou být zobrazené informace dostačující, je možné si zobrazit okno s detailnějšími informacemi o záznamu, které obsahuje několik položek navíc a také vysvětlení činnosti, kterou se program snažil nad objektem vykonat, a popis třídy objektu.

Jako obvykle je dobré si ukázat, jak takové okno vypadá – to je možné vidět na obr. 5.7.



Obr. 5.7: Detail AVC zprávy

Toto okno má pouze informativní charakter a lze z něj vyčíst pouze základní údaje o jednotlivé AVC zprávě. Veškeré zpracování se provádí opět z hlavního okna programu.

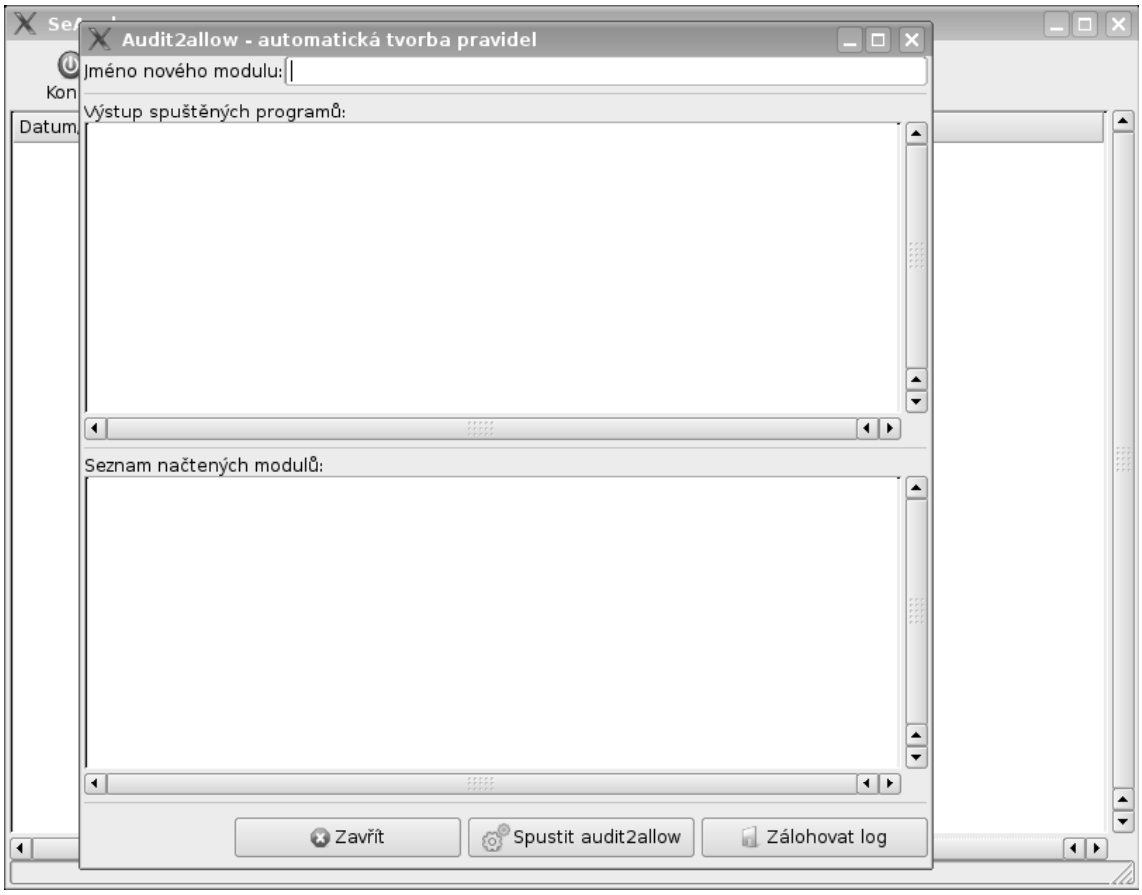
5.6.4 Zpracování záznamů pomocí Audit2allow

Nejdříve se seznámíme se způsobem vytvoření nových modulů a pravidel pomocí utility nazvané `audit2allow`. Již při nastavení jsme měli možnost zvolit mezi plně automatickým režimem a volbou poloautomatického zpracování, kdy jsou uživateli ukázána generovaná pravidla a má možnost před kompilací je změnit či upravit.

Plně automatické zpracování

Po vybrání tlačítka „Audit2allow“ v horní listě a při nastavené volbě automatického zpracování je uživateli ukázáno nové okno, které se skládá ze několika částí:

- vstupní pole pro název nového modulu
- výstup spouštěných programů na serveru
- výpis načtených modulů
- funkční tlačítka



Obr. 5.8: Automatické zpracování audit2allow – prázdný formulář

Vzhled tohoto okna ihned po zobrazení je vidět na obr. 5.8.

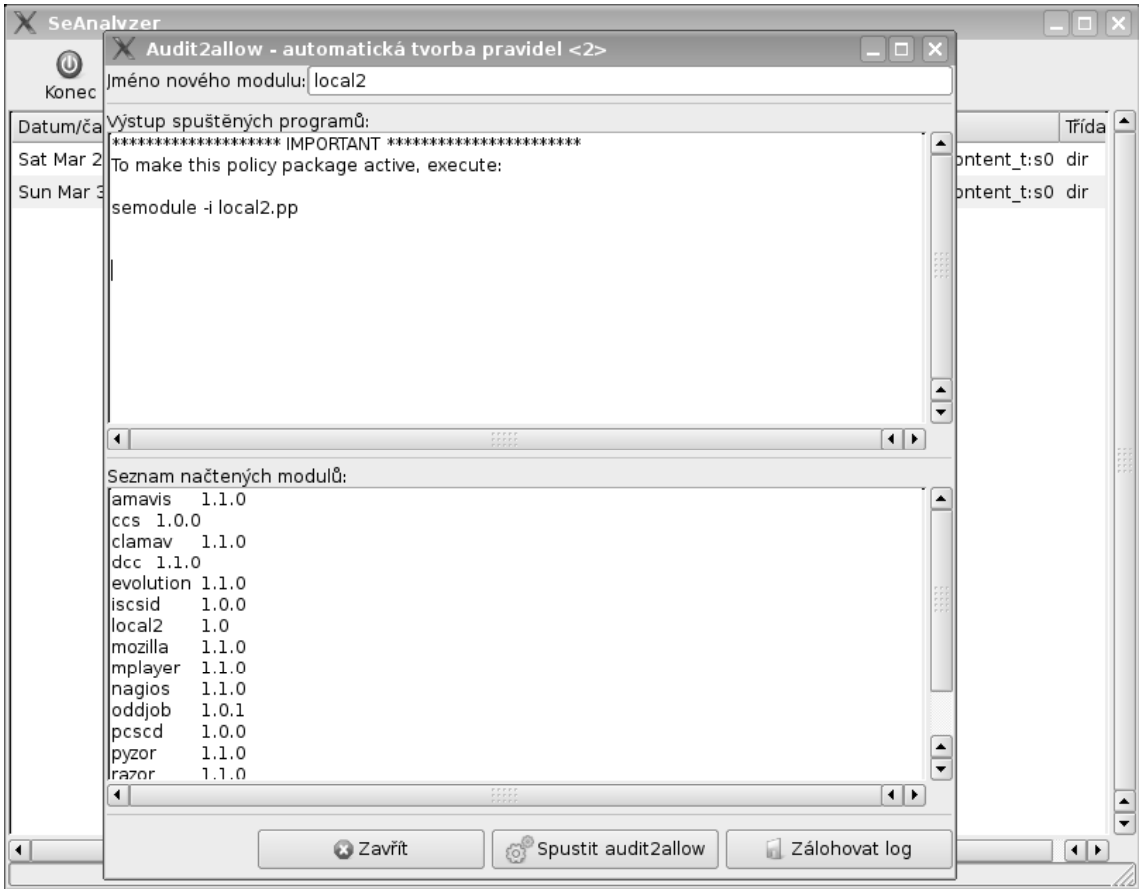
Do horního vstupního pole se zadává název nového modulu. Vzhledem k tomu, že tvoříme pravidla automaticky a tudíž pro více programů, je dobré vytvářet moduly pojmenované například ve tvaru *local-<časová značka>* či *auto-<časová značka>*. Protože pokud bychom použili název nějakého programu a později k němu tvořili pravidla ručně, mohl by pomalu ale jistě vzniknout zmatek, které moduly patří ke které aplikaci.

Po vyplnění názvu nového modulu pravidel lze přistoupit k vlastnímu spuštění automatického nástroje. Jak tato činnost probíhá již bylo popsáno v kapitole 4.5.2, přesto neuškodí si tento postup zopakovat. Celý proces spočívá ve spuštění jediného příkazu na SSH serveru, a to:

```
cat /var/log/audit/audit.log | audit2allow -M nazev_modulu
```

Pokud vše proběhne hladce, měla by se ve výpise spuštěných programů objevit informace o kompilaci a nahrání modulu do jádra SELinuxu. Ve spodním textovém poli by měl být zobrazen seznam aktuálně nahraných modulů a mezi nimi by měl

figurovat i nově vytvořený. Úspěšný pokus je vidět na obr. 5.9. Ihned poté je spuštěno přeznačení celého souborového systému – tento příkaz je spuštěn na vzdáleném počítači na pozadí, takže by neměl blokovat činnost aplikace.



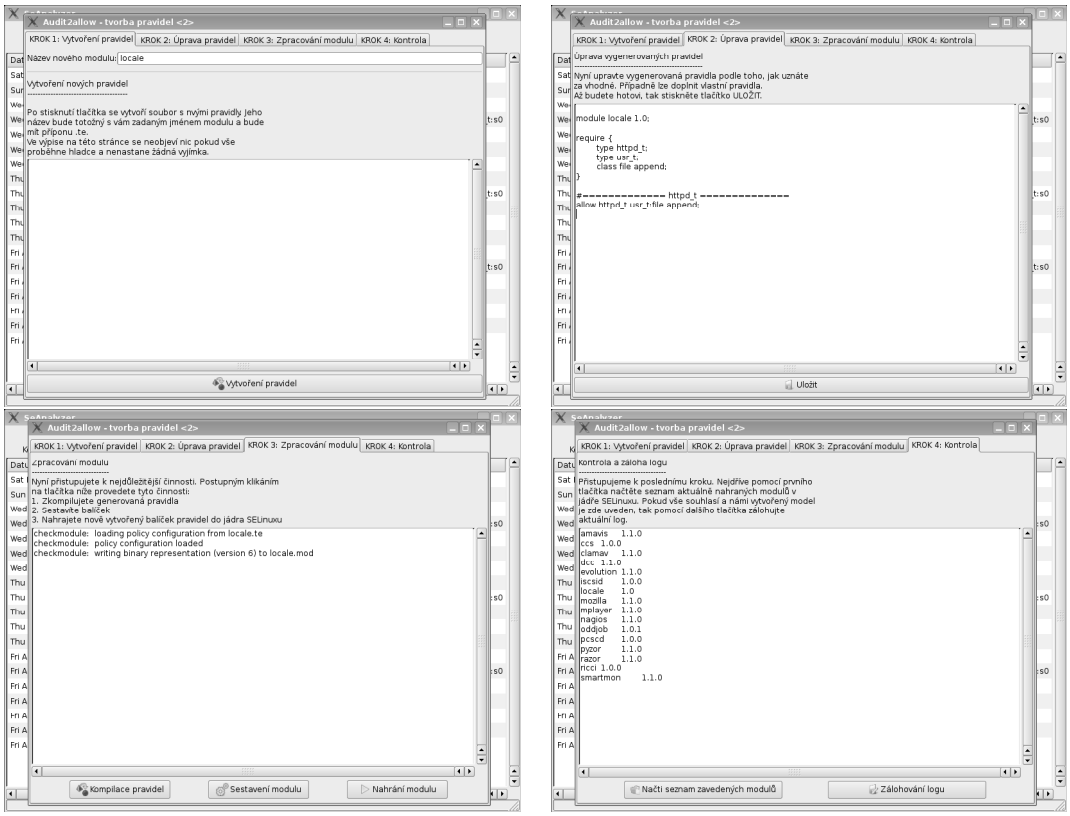
Obr. 5.9: Automatické zpracování audit2allow – úspěšné provedení

Pokud vše proběhne v pořádku, je vhodné použít třetí dostupné tlačítko „Zálohovat log“ a tím zajistit, že se nebudou generovat znovu stejná pravidla. Po stisku tlačítka je soubor `/var/log/audit/audit.log` překopírován do stejného adresáře s názvem podle vzoru `audit.log-RRRRMMDD_HHMMSS` a obsah původního souboru je vymazán.

Poloautomatická tvorba modulu

Pokud se rozhodneme pro použití utility `Audit2allow`, je lépe používat právě tzv. poloautomatickou tvorbu pravidel. Jak již bylo zmíněno dříve, tímto způsobem se vytvoří nová pravidla, avšak před samotnou kompilací modulu má uživatel možnost generovaná pravidla upravit nebo doplnit.

Tvorba pravidel tímto způsobem začíná tím, že si v nastavení programu zvolíme poloautomatickou (manuální) tvorbu pomocí programu Audit2allow. Poté se



Obr. 5.10: Poloautomatická tvorba pomocí audit2allow

klepnutím na příslušné tlačítko zobrazí okno skládající se z několika záložek, které představují jednotlivé kroky tvorby nového modulu:

1. **Vytvoření pravidel** – v této části je uživatel dotázán na zadání jména nového modulu a pomocí tlačítka může vytvořit nová pravidla.

2. **Úprava pravidel** – do textového pole v této záložce je automaticky nahrán nově generovaný soubor .te, obsahující pravidla vytvořená pomocí utility audit2allow. Po případné úpravě je nutné soubor pomocí tlačítka uložit.

3. **Zpracování modulu** – v této záložce probíhá zřejmě nejvíce činností, ale pro uživatele nejsou náročné. Nejdříve je nutné nová pravidla zkompilovat, poté z nich sestavit modul a následně jej nahrát do jádra SELinuxu. Pokud vše proběhne dobře, lze přistoupit k poslední záložce.

4. **Kontrola** – v tomto kroku již probíhá pouze základní kontrola správnosti nahrání nového modulu do jádra a dále je zde umístěno tlačítko pro zálohu aktuálního záznamového souboru. Při záloze záznamového souboru rovněž dojde k přeznačení systému souborů.

Jestliže vše proběhlo v pořádku, měl by být v jádře SELinuxu nahrán nově vytvořený modul a pravidla by měla být vytvořena tak, aby byl systém ochraňován i s ohledem na nově instalované programy, které nejspíše způsobovaly výjimky.

Vytvoření pravidel v těchto čtyřech krocích je možné vidět na obrázku 5.10.

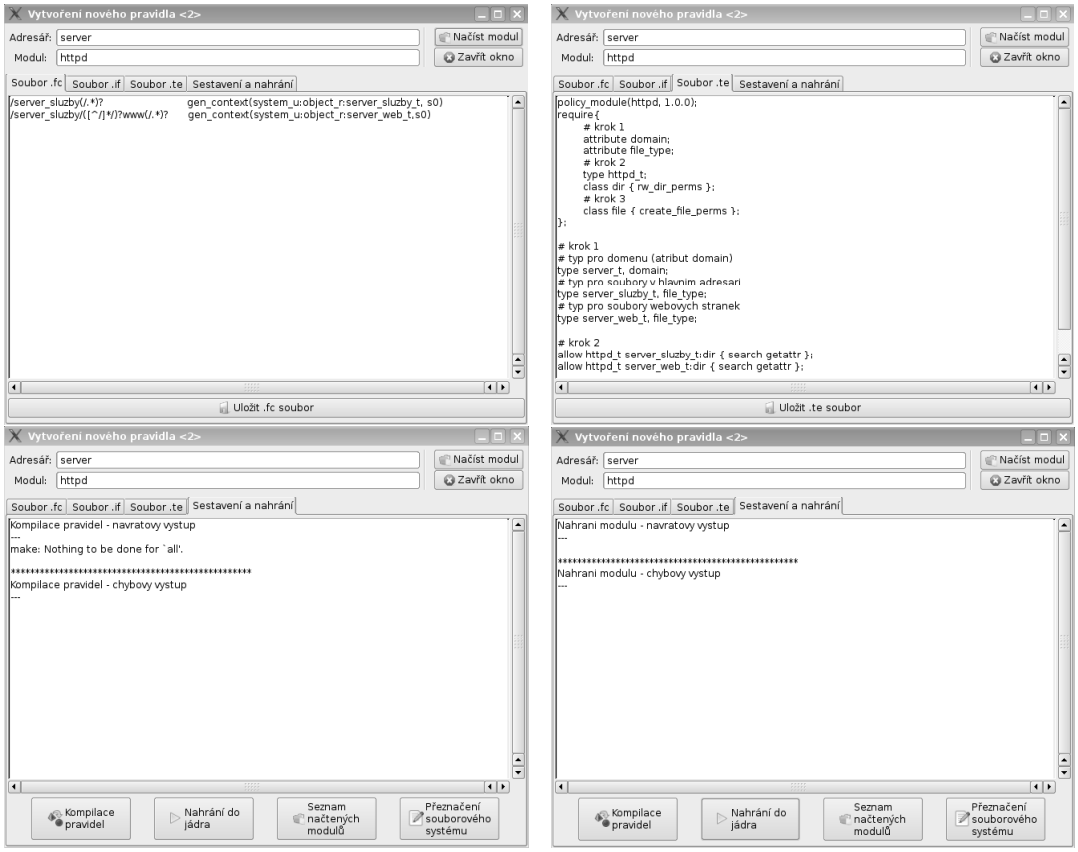
5.6.5 Ruční psaní nového pravidla

Aplikace také nabízí jednoduchý formulář pro psaní vlastních pravidel. Po klepnutí na tlačítko „Vytvořit pravidlo“ se uživateli objeví formulář, který je zobrazen na obrázku 5.11.

Obr. 5.11: Formulář pro ruční psaní pravidel

Celý systém psaní nového pravidla začíná zadáním názvu podadresáře ve zdrojových souborech SELinuxu a následně jména nového modulu. Pokud upravujeme již definovaný modul, pomocí tlačítka v pravém horním rohu načteme obsahy příslušných souborů ze serveru a zobrazíme je v příslušných polích formuláře.

Vlastní tvorba se provádí ve čtyřech záložkách. V prvních třech se postupně vyplňují soubory typu .fc, .if a .te. Po jejich vypsání je samozřejmě potřeba každý soubor uložit pomocí tlačítka ve spodní části záložky.



Obr. 5.12: Shrnutí ruční tvorby pravidel

Pokud již máme vyplněny základní soubory, dostaneme se ke čtvrté záložce, kde je potřeba postupně provést několik činností:

1. zkompilovat nově vytvořená pravidla do balíčku s příponou .pp
2. nahrát nově vytvořený modul do jádra SELinuxu
3. proběhne-li vše bez chyby, můžeme si načíst seznam aktuálně nahraných modulů v jádře SELinuxu a zkontrolovat, zda je mezi nimi i námi nově vytvořený balíček
4. v případě, že jsme shledali vše v pořádku, je nezbytné ještě přeznačit systém souborů, tak aby se nově definovaná pravidla v systému projevila

Pokud během žádného kroku nenastala výjimka, měla by námi vytvořená pravidla pomocí nového modulu v systému fungovat. V případě, že v některém z kroků

dojde k chybě, je potřeba upravit soubory s pravidly a znovu provést všechny kroky v poslední záložce.

Celý proces ruční tvorby je shrnut na obrázku 5.12

5.7 Funkční rozbor aplikace

V následujících odstavcích popíšeme části zdrojových kódu a jejich použití v aplikaci. Nejdříve však začneme základní částí uživatelské přívětivosti – tvorbou grafického vzhledu. Poté se jen zběžně seznámíme se základním konceptem aplikace. Zdrojové kódy je možné nalézt v elektronické příloze této práce.

5.7.1 Tvorba grafického vzhledu

Ruční psaní grafického vzhledu programu by bylo poměrně zdlouhavé a hlavně neefektivní. Proto existují specializované programy, které nám usnadňují tvorbu vzhledu.

Pro zvolenou grafickou knihovnu dobře poslouží program s názvem *Glade*, který sice není primárně určen k tvorbě vzhledu pro programovací jazyk Python, ale v knihovně PyGTK existují funkce, které s takto definovaným vzhledem umí pracovat.

Každý formulář vykreslíme do samostatného souboru, tudíž nám vznikne šest souborů s různými formuláři – základní okno programu, nastavení aplikace, detailní okno jednoho záznamu, plně automatická a poloautomatická tvorba pomocí audit2allow, manuální tvorba nových pravidel. Tyto soubory se vzhledy jednotlivých oken jsou uloženy v adresáři `seanalyzer`.

5.7.2 Základní soubor programu

Základním souborem, který je také spouštěn, je `seanalyzer.py`. Po spuštění se vykoná konstruktor spouštěcí třídy, v něm je v první řadě načten grafický vzhled hlavního okna a toto je zobrazeno. Poté jsou jednotlivým tlačítkům přiřazeny definice funkcí, tak aby byly správně ošetřeny.

Pro každou funkci (a tudíž pro každé nové okno) je volána zvláštní třída, která ve svém konstruktoru vytváří okna a přiřazuje tlačítkům akce. Těmito jsou pak volány jednotlivé třídy a jejich příslušné funkce, které jsou umístěny v některém z balíčků.

Aplikace je rozdělena na dva různé balíčky – jeden z nich ošetřuje nastavení programu a druhý veškerou činnost, která je spojena se zpracováním záznamů a tvorbou nových pravidel. Tyto balíčky jsou umístěny v adresářích `analyze` a `configure`.

5.7.3 Balíček pro uložení konfigurace

Mimo soubor konstruktoru je v tomto adresáři jediný skript `userconf.py`, obsahující funkci, která otevře konfigurační soubor a přepíše jeho obsah novými hodnotami, jež byly získány z nastavovacího formuláře.

5.7.4 Balíček analýzy a zpracování záznamů

Druhým a nejdůležitějším je balíček funkcí, které zajišťují činnost spojenou se zpracováním záznamů SELinuxu, a dále funkce pro tvorbu nových pravidel. Veškeré soubory se nacházejí v adresáři `analyze` a jedná se hlavně o tyto tři:

1. **audit2allow.py** – jak již název napovídá, jedná se o soubor obsahující definice funkcí sloužících k tvorbě nových pravidel pomocí utility audit2allow. Na začátku každé z nich je volána funkce, která zajišťuje načtení nastavení ze souboru a navrácení těchto hodnot. Poté již mohou probíhat příslušné činnosti i s ohledem na použitou distribuci Linuxu.

2. **createmodule.py** – zde jsou obsaženy funkce pro ruční psaní pravidel a jejich následné zpracování a nahrání do jádra SELinuxu. Na začátku je také uvedena funkce zajišťující zpracování souboru s nastavením a je volána z každé z funkcí, tak aby bylo zajištěno, že bude program pracovat ve správném adresáři.

3. **nactissh.py** – tento soubor obsahuje pouze jednu funkci, která nejdříve načte nastavení a poté se pokusí připojit na síťový server a zpracovat záznamový soubor pomocí regulárního výrazu. Výsledek hodnot, které se s výrazem shodují, uloží do pole, jež je vráceno zpět do hlavní části aplikace.

5.7.5 Doplňující informace

V detailním zobrazení záznamu jsou ve spodní části blíže specifikovány některé části záznamu – zpráva, oprávnění a třída objektu. Informace o nich jsou načítány z příslušných souborů v adresáři `info`. Jsou v obyčejném textovém formátu, kdy obsahují jednotlivé prvky oddělené dvojtečkou – skládají se z názvu, anglického a českého popisu. Zjednodušenou formu s názvem a českým popisem je možné najít v přílohách této práce.

6 POUŽITÍ DEFINICE PRAVIDEL

Nyní se pokusme ukázat praktické zabezpečení pomocí nově psaných pravidel. Půjde o zabezpečení virtuálních domén webového serveru Apache. Pravidla pro něj jsou podle očekávání velmi dobře napsána, poněvadž jde o téměř nejpoužívanějšího démona. My si vytvoříme pravidla s novými typy speciálně pro ukázku tvorby nových pravidel a postupné povolování pomocí dalších TE (Type Enforcement – Vynucení typu) pravidel.

Praktická ukázka tvorby bude provedena na operačním systému společnosti RedHat, a to konkrétně na jeho bezplatné verzi CentOS ve verzi 5. Použitelná je samozřejmě i na jiných systémech, je však potřeba počítat s tím, že konfigurační soubory mohou být umístěny v jiném adresáři.

6.1 Zabezpečení serveru Apache

Veškerý obsah virtuálních domén oddělíme od standardního systému tím, že si vytvoříme zvláštní adresář pouze pro náš případ, který určitě nebude ošetřen v žádné definici pravidel.

Proto na serveru vytvoříme adresář `/server_sluzby` a do něj příslušné podadresáře značící virtuální domény. Pro démona Apache poté vytvoříme další podadresář již standardního jména `www`, kam se budou ukládat stránky. Celá cesta tedy bude vypadat například následovně:

```
/server_sluzby/example.com/www
/server_sluzby/mojedomena.cz/www
```

Nyní si projdeme v několika krocích samotné vytvoření podmínek pro ukázku psaní nového pravidla:

1. vytvoření adresářové struktury
2. editace a vytvoření nových virtuálních domén serveru Apache
3. napsání příslušných pravidel, přeznačení souborového systému a postupné testování správnosti pravidel

6.1.1 Vytvoření adresářové struktury

Ze všeho nejdříve je potřeba na testovacím serveru vytvořit příslušnou strukturu adresářů, která nám bude simulovat reálný server. Jako první vytvoříme hlavní adresář, kam se budou ukládat data domén, a poté příslušné podadresáře pro jednotlivé

domény a adresář pro webové stránky. Jako název testovacích domén si zvolme třeba *domena1.cz* a *domena2.cz*.

Sestava příkazů pak bude vypadat takto (spouštěno pod uživatelem root):

```
$ mkdir /server_sluzby/

$ mkdir /server_sluzby/domena1.cz/
$ mkdir /server_sluzby/domena1.cz/www

$ mkdir /server_sluzby/domena2.cz/
$ mkdir /server_sluzby/domena2.cz/www
```

Toto by mělo být pro ukázku dostačující.

6.1.2 Vytvoření virtuálních domén

Vytváření virtuálních domén probíhá v adresáři s nastavením serveru Apache. Veškeré konfigurační soubory se v distribuci CentOS nacházejí v `/etc/httpd`. Pro jednoduchost budeme psát definici virtuálních domén do hlavního konfiguračního souboru `httpd.conf`, který je umístěn v podadresáři `conf`. Budeme vytvářet dvě virtuální domény s názvy *domena1.cz* a *domena2.cz*. Přesným psaním definice domén se na tomto místě zabývat nebudeme, neboť čtenář by měl být aspoň základně s tímto seznámen. Na tomto místě uvedu pouze samotný výpis nastavení jedné z domén.

Celý postup příkazů lze shrnout tímto výpisem:

```
$ cd /etc/httpd/conf/
$ vi httpd.conf
```

Níže je uveden výpis velmi jednoduché definice virtuálního serveru pro doménu domena1.cz a domena2.cz. Tyto definice se většinou píší na závěr konfiguračního souboru.

```
NameVirtualHost *:80

<VirtualHost *:80>
   DocumentRoot /server_sluzby/domena1.cz/www
   ServerName domena1.cz
</VirtualHost>

<VirtualHost *:80>
   DocumentRoot /server_sluzby/domena2.cz/www
```

```
    ServerName domena2.cz
</VirtualHost>
```

Po změně nastavení je potřeba démona Apache restartovat, aby mohl přijmout novou konfiguraci.

Nyní je ještě potřeba vytvořit aspoň jednoduchou stránku v adresářích, kam budou směřovat jednotlivé domény. Lze vytvořit třeba soubor `index.html` s následujícím obsahem, který plně postačuje pro otestování:

```
<h1>domena[1/2].cz</h1>
```

Ještě je potřeba hostitelskému počítači „říci“, kam má směřovat naše domény. Toto nastavení se provádí v souboru `/etc/hosts` a to přidáním jednoho řádku:

```
# IP     hostname.hostdomain
172.16.94.130     domena1.cz domena2.cz
```

6.1.3 Vytvoření pravidel

Nyní přistoupíme k nejdůležitější části – psaní pravidel pro náš ukázkový příklad. Nejdříve navrhneme nové bezpečnostní kontexty (a to hlavně nové typy) skládající se z definice `uživatel:role:typ`, které jsou blíže popsány v části 4.2.

Volbu bezpečnostních kontextů začněme pěkně popořadě, tedy od hlavního adresáře a jeho bezprostředního obsahu. Tento bude mít kontext systémového uživatele s typem *server_sluzby_t*:

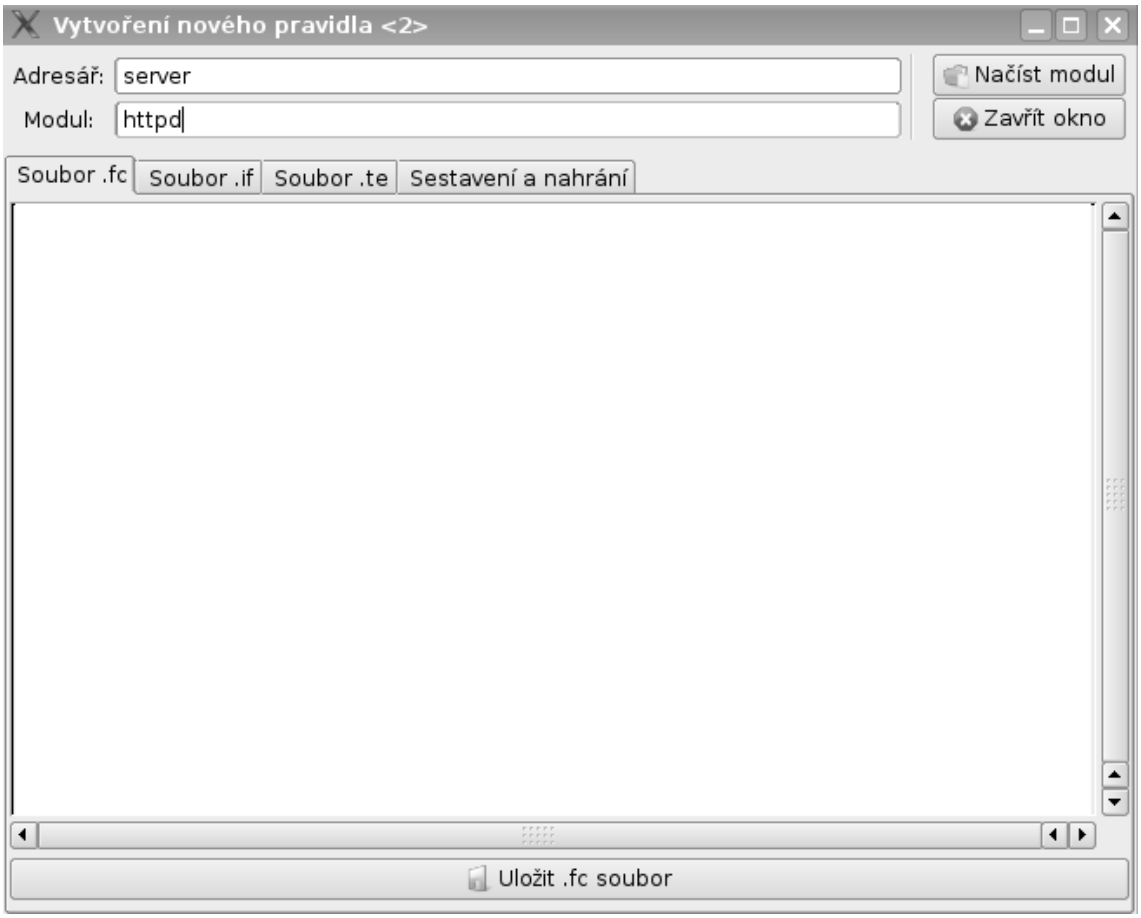
```
system_u:object_r:server_sluzby_t
```

Nyní zvolme nový typ pro adresáře s obsahem webových stránek:

```
system_u:object_r:server_web_t
```

Toto nám pro začátek stačí a můžeme tedy přistoupit k postupnému vymýšlení souborů .fc a .te, jejichž základní popis je popsán v kapitole 4.5.1. Soubor s definicí maker (.if) se příliš nevyužívá a v našem případě jej také nebudeme definovat.

Pusťme se do manuální definice pravidel – vše začněme kliknutím na tlačítko „Vytvořit pravidlo“ na hlavní tlačítkové liště. Objeví se okno zobrazené na obrázku 6.1. Více o tom, kde v tomto okně definovat jednotlivé prvky, je možné najít v části 5.6.5. Pouze připomeňme, že se začíná zadáním jména adresáře, kam chceme modul uložit, a názvu samotného modulu. V případě, že jde o opravu již definovaného modulu, použije se tlačítko pro načtení, které vyplní jednotlivé záložky obsahem příslušných souborů.



Obr. 6.1: Formulář pro manuální tvorbu pravidla

Definice značení systému (soubor .fc)

V tomto souboru připojíme jednotlivým adresářům a jejich obsahu námi vymyšlené bezpečnostní kontexty. Ve výpise je vidět postupně pravidlo značení pro hlavní adresář a adresář s obsahem webových stránek:

```
# hlavní adresář
/server_sluzby(/.*)?            gen_context(system_u:object_r:
                                            server_sluzby_t, s0)
# adresář s~webovými stránkami
/server_sluzby/([^/]*/)?www(/.*)?  gen_context(system_u:object_r:
                                              server_web_t,s0)
```

Tento text zapíšeme do příslušné záložky a dáme uložit soubor s příponou fc. Celá definice patří na jeden řádek, zde musí být ukázka zalomena na více řádků.

Vlastní TE pravidla

Teď přijde asi nejtěžší část vytváření pravidel. Jejich psaní bývá často zkoušením a postupným povolováním dalších podle toho, co se ukáže v záznamech SELinuxu jako zablokované (denied). Jsou v podstatě 2 možnosti jak to zkoušet – ve vynucovacím módu (*enforcing* mód), kdy se nám pravidla hned projevují, nebo v *permissive* módu, kdy se pouze zapisují záznamy a nic se nevynucuje.

Na začátek zapíšeme hlavičku modulu. Pak v části *require* načteme všechny již definované atributy, typy, třídy či makra. Pro začátek načteme typy pro soubor (file_type):

```
policy_module(server, 1.0.0);
require{
   attribute file_type; # atribut pro soubory, adresáře
};
```

Dále je potřeba přiřadit atribut souborových typů (file_type) všem námi definovaným:

```
# typ pro soubory v~hlavním adresáři
type server_sluzby_t, file_type;

# typ pro soubory s~web stránkami
type server_web_t, file_type;
```

Všechen text uvedený ve výpisech výše přepíšeme do příslušné záložky formuláře a samozřejmě uložíme změněný soubor.

Sestavení a nahrání pravidel

Po definici a jakékoliv změně je potřeba pravidla znovu zkompilovat pomocí příkazu `make -f /usr/share/selinux/devel/Makefile` a nahrát novou verzi modulu do jádra SELinuxu – všechna tato činnost se provádí v poslední záložce formuláře. Pro každou část zpracování nově definovaného modulu je určeno samostatné tlačítko. Ve výpise se vždy objeví návratové hodnoty a případné chybové hlášky, které nastaly při spuštění. Mezi tlačítky nechybí ani kontrola, zda se modul nahrál do jádra SELinuxu v pořádku – slouží k tomu výpis aktuálně přítomným modulů v jádře. V neposlední řadě je také potřeba přeznačení systému, které lze provést pomocí posledního tlačítka – tímto se přeznačí celý souborový systém, takže tato činnost může nějakou chvíli trvat.

Kontrola správnosti

Nyní zkusme v prohlížeči načíst například stránku http://domena1.cz/. Pro mnohé může být překvapením, že se objeví stránka oznamující zamezený přístup. Teď přichází na řadu zkoumání záznamů SELinuxu, abychom zjistili, co je třeba ještě povolit. Proto ve správně nastaveném programu načteme data ze serveru. Měli bychom obdržet výpis zobrazený na obrázku 6.2.

Jak je vidět, ve výpise se nám objeví jedna či více zpráv patřící ke kontextu `system_u:object_r:server_sluzby_t`. V podstatě by se měly objevit 2 druhy zpráv – jeden o zamezení procházení adresáře (search) a druhý o získání atributů (getattr). Detail těchto 2 zpráv je vidět na obrázku 6.3. Obojí zprávy se týkají adresáře, což lze usoudit ze třídy *dir*.

Úprava pravidel

Pokud nastala nějaká výjimka, přichází na řadu úprava pravidel. Ve formuláři se vyplní název adresáře a jméno modulu a pomocí tlačítka „Načíst modul“ jsou získána data ze souborů přímo na serveru a příslušná pole formuláře vyplněna.

Teď je potřeba podle získaných poznatků přidat další pravidla – v našem případě do souboru .te. Ze zdrojového kontextu je jisté, že je potřeba ještě do části require začlenit typ *httpd_t*. V cílovém kontextu je námi definovaný typ *server_sluzby_t*, který tedy není potřeba znovu vkládat. Dále třeba „načíst“ právě ze třídy *dir* operace pro procházení adresáře a získávání atributů. Tudíž do části require doplníme:

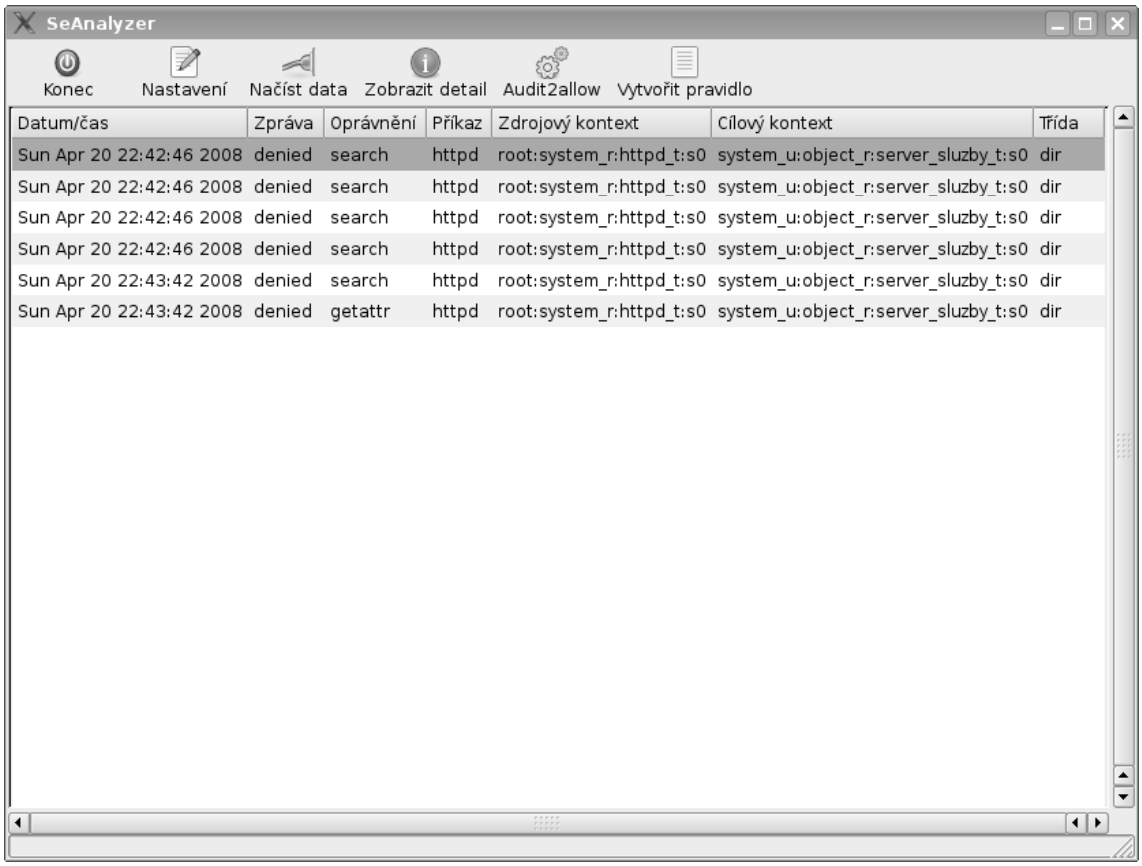
```
type httpd_t;
class dir { search getattr };
```

A již nezbývá nic jiného, než povolit přechod z typu *httpd_t* na *server_sluzby_t* pro adresářovou třídu a její zakázané operace. Jelikož jsme definovali dva nové typy, je předpoklad, že totéž bude potřeba povolit pro oba:

```
allow httpd_t server_sluzby_t:dir { search getattr };
allow httpd_t server_web_t:dir { search getattr };
```

Nezapomeňme na to, že je potřeba znovu zkompilovat pravidla a zavést jejich novou verzi do jádra SELinuxu. Pokud jsme neměnili soubor s příponou fc, není již potřeba přeznačení systému souborů. V opačném případě se této činnosti nevyhneme.

Nyní se opět vrátíme k metodě pokus-omyl a znovu se pokusíme otevřít stránku jedné z námi definovaných virtuálních domén. Očekávání, že již půjde vše dobře, by bylo předčasné. Nezbývá než si znovu načíst záznamy SELinuxu. Měl by se objevit minimálně jeden další záznam, jehož výpis a detail je vidět na obrázku 6.4.



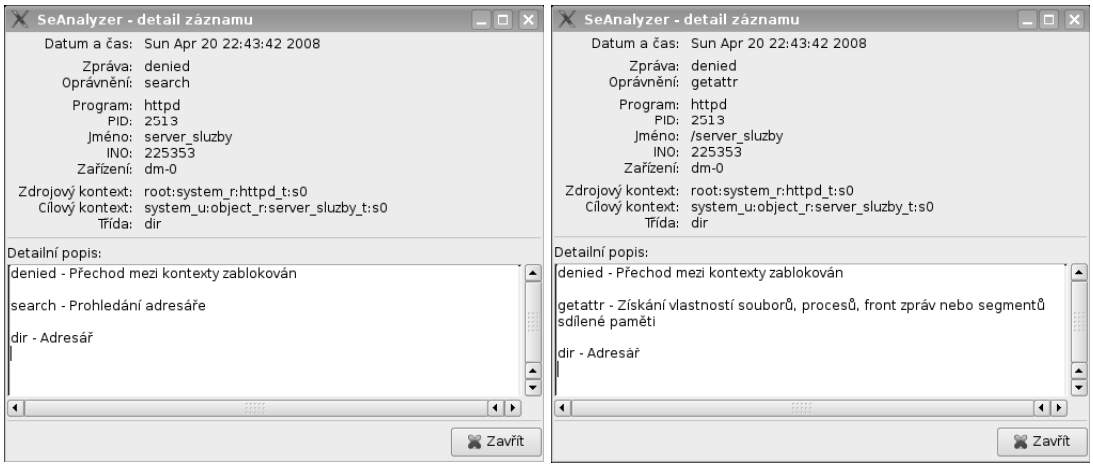
Obr. 6.2: Výpis po definici značení a souborových typů

Na první pohled by mělo být jasné, že již není problém s třídou obsluhující adresáře, ale nyní se soubory. Je to logické, protože jsme ještě nad nimi nepovolili žádné operace. Takže se do toho pusťme – jako vždy nejdříve je potřeba načíst příslušnou třídu a operace v ní. Pro soubory lze načíst celou sadu operací *create_file_perms*, která má pod sebou všechny základní operace, jež lze se soubory provádět. Tedy do části require je potřeba doplnit použití třídy:

```
class file { create_file_perms };
```

A teď vytvořme vlastní pravidlo. Z výpisu by se nabízelo povolit pouze operaci *getattr*, protože ta jediná zatím způsobila výjimku. Ale zamysleme se nyní nad tím, co vše vlastně se soubory budeme dělat – budeme přeci také chtít soubory v tomto adresáři číst (*read*), zapisovat do nich (*write*), vytvářet (*create*), přejmenovávat (*rename*) nebo také mazat (*unlink*). Proto povolíme všechny tyto operace jedním příkazem:

```
allow httpd_t server_web_t:file { getattr read write create
                                  rename unlink };
```



Obr. 6.3: Detail zamezení procházení a získání atributů

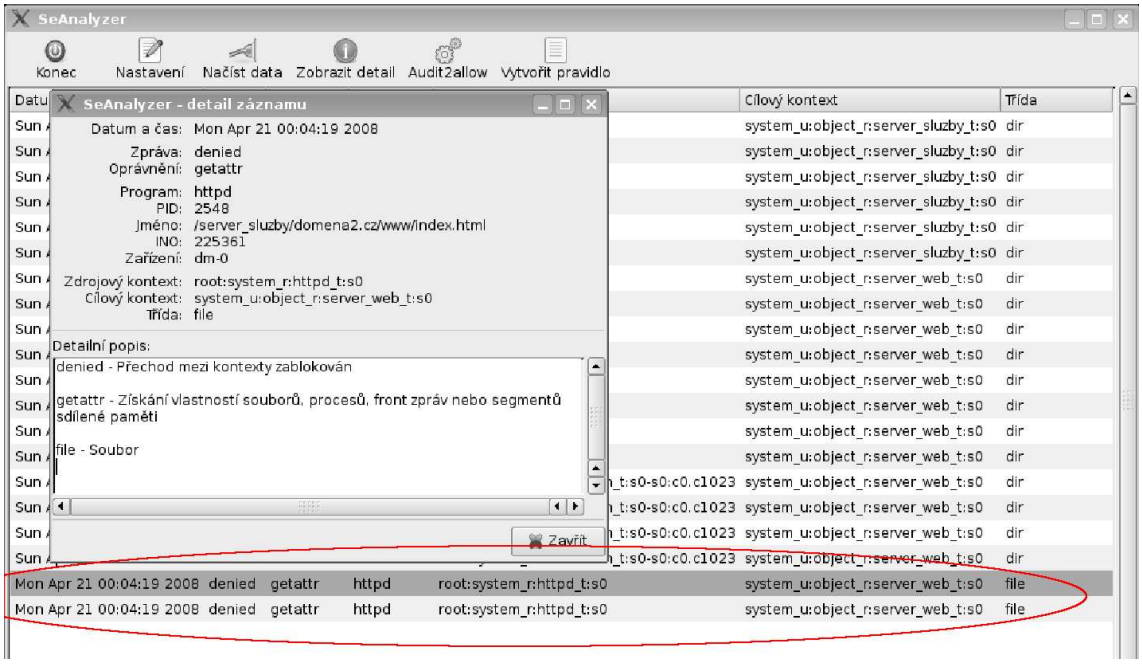
Nyní by se měla objevit námi vytvořená stránka oznamující, na které doméně se nacházíme. Mohou se samozřejmě objevit i další chyby, vše záleží na systému a na verzi standardních pravidel, ale pokud si projdeme právě vytvářená pravidla, není problém přidávat další.

Nyní uveďme celkový výpis toho, co jsme v této části nadefinovali:

```
policy_module(server, 1.0.0);
require{
   # krok 1
   attribute domain;
   attribute file_type;
   # krok 2
   type httpd_t;
   class dir { rw_dir_perms };
   # krok 3
   class file { create_file_perms };
};

# krok 1
# typ pro domenu (atribut domain)
type server_t, domain;
# typ pro soubory v~hlavnim adresari
type server_sluzby_t, file_type;
# typ pro soubory webovych stranek
type server_web_t, file_type;

# krok 2
```



Obr. 6.4: Po přidání práv pro adresáře

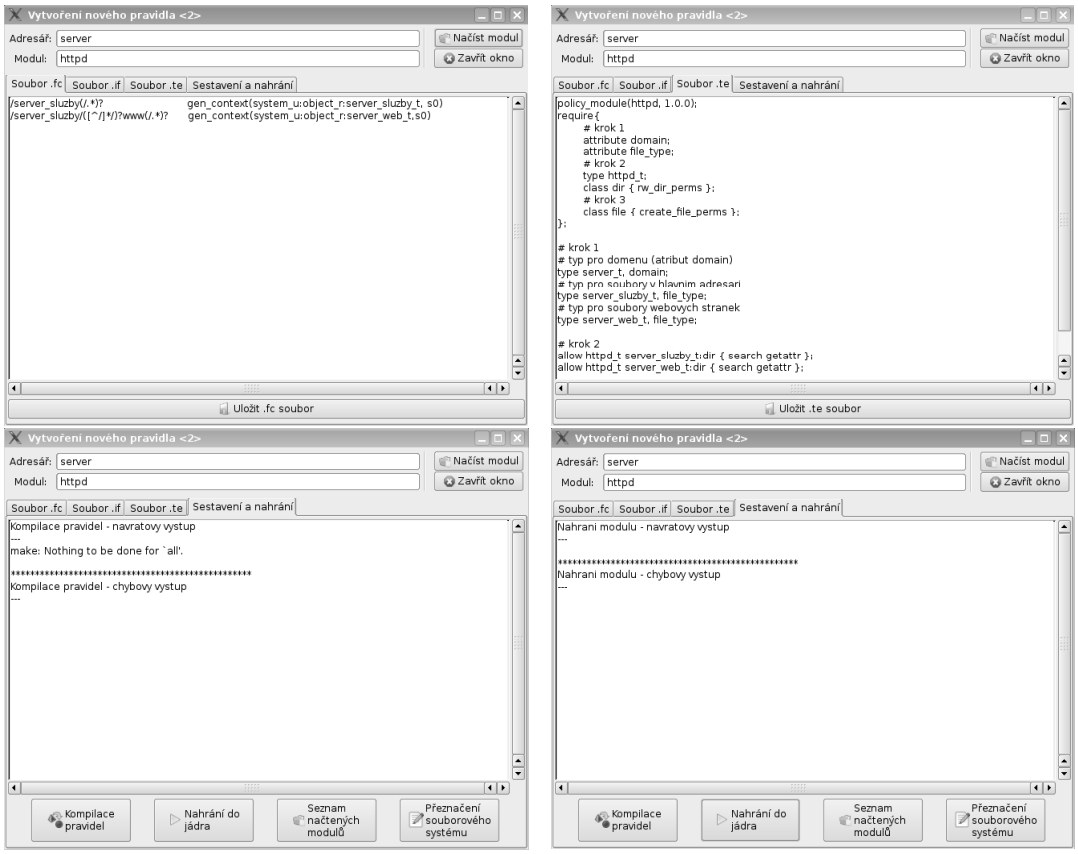
```
allow httpd_t server_sluzby_t:dir { search getattr };
allow httpd_t server_web_t:dir { search getattr };

# krok 3
allow httpd_t server_web_t:file { getattr read write create
                                  rename unlink };
```

Pro představu o jednotlivých krocích tvorby je určen obrázek 6.5, kde jsou zobrazeny čtyři základní částí tvorby nového či úpravy stávajícího pravidla i s případnými výpisy. Není zde uvedena kontrola pomocí seznamu načtených modulů a přeznačení systému souborů, které nemá většinou žádnou návratovou hodnotu.

6.2 Spouštění skriptů pod webovým serverem

Nyní si prakticky můžeme ověřit to, co jsme dělali v předchozí části. Na webových stránkách je také možné spouštět tzv. CGI skripty – jde o programy, které mohou vykonávat přímo příkazy operačního systému. Z tohoto důvodu je také jejich použití bezpečnostním rizikem, a proto není většinou jednotlivým uživatelům dovoleno spouštět je z normálního adresáře, ale jsou umístěny do speciální složky. V této části si zkusme tuto složku zabezpečit – nejdříve teoreticky a poté prakticky.



Obr. 6.5: Manuální tvorba SELinux modulu ve čtyřech obrazech

6.2.1 Teoretické zabezpečení složky s CGI skripty

Nejprve si nadefinujme nový bezpečnostní kontext přímo pro námi zvolený adresář `/server_sluzby/cgi-bin/`, kam budeme CGI skripty ukládat a spouštět. Podle již použitého vzoru užijme kontext s typem *server_script_t*:

```
system_u:object_r:server_script_t
```

V souboru definujícím pravidla TE je potřeba nejdříve načíst atributy pro doménu, adresář a soubory. Dále je nutné začlenit určité typy, které se určitě budou vyskytovat – zde předpokládáme pouze jeden typ a to `httpd_t`. Pro třídu adresářů budeme chtít jak procházení, tak získávání atributů – vše lze získat z třídy operací `rw_dir_perms`. U souborů budeme chtít také základní operace, stejně jako jsme je použili v předchozí části, a navíc bude potřeba zajistit, aby se soubory mohly spouštět.

Tyto úvahy by měly být teoreticky dostačující, zkusme tedy přistoupit k praktickému sepsání nového modulu.

6.2.2 Praktické zabezpečení CGI skriptů

Nová pravidla definujme ve stejném adresáři jako předchozí, pouze si vytvořme novou trojici souborů mající tvar například `httpd_cgi.[fc,if,te]` – do horních polí formuláře vyplňme název adresáře `server` (ať máme podobná pravidla v jednom adresáři) a název modulu zvolme `httpd_cgi`. Formulářové pole pro soubor .fc vyplňme pravidly pro značení adresářů podle úvahy a nezapomeňme jej uložit.

V teoretickém rozboru není nic o vlastních makrech, tudíž soubor .if opět nevyužijeme.

Vyplňme tedy formulář pro soubor `httpd_cgi.te` – doplňme pravidla pro povolení uvažovaných přechodů a operací. Pokud správně uvažujeme, měli bychom dostat něco podobného následujícímu výpisu:

```
# soubor: server_cgi.fc
/server_sluzby/cgi-bin(/.*)?  gen_context(system_u:object_r:
                                          server_script_t,s0)

# soubor: server_cgi.te
policy_module(httpdcgi, 1.0.0);

require{
   attribute exec_type;
   attribufe file_type;
   type httpd_t;
   class dir { rw_dir_perms };
   class file { create_file_perms };
};

# typ pro soubory v~adresari
type server_script_t, file_type, exec_type;

allow httpd_t server_script_t:dir { search getattr };
allow httpd_t server_script_t:file { getattr read write create
                                     rename unlink execute };
```

Zkusme tedy takto vytvořená pravidla zkompilovat, načíst do jádra SELinuxu a přeznačit souborový systém – to vše provedeme v již používané poslední záložce formuláře.

Kontrola správnosti pravidel

Pro to, abychom mohli zkontrolovat, zda jsme pravidla napsali na poprvé dobře, je potřeba udělat ještě pár drobných úprav přímo na serveru.

V adresáři cgi-bin vytvořme soubor *test.cgi*, který bude mít jednoduchý obsah (zobrazí se formátované přivítání):

```
#!/usr/bin/perl

print "Content-type: text/html\n\n";
print <<HTML;
   <html>
      <head>
         <title>CGI Test</title>
      </head>
      <body>
         <h1>CGI:</h1>
         <p>ahoj</p>
      </body>
HTML
exit;
```

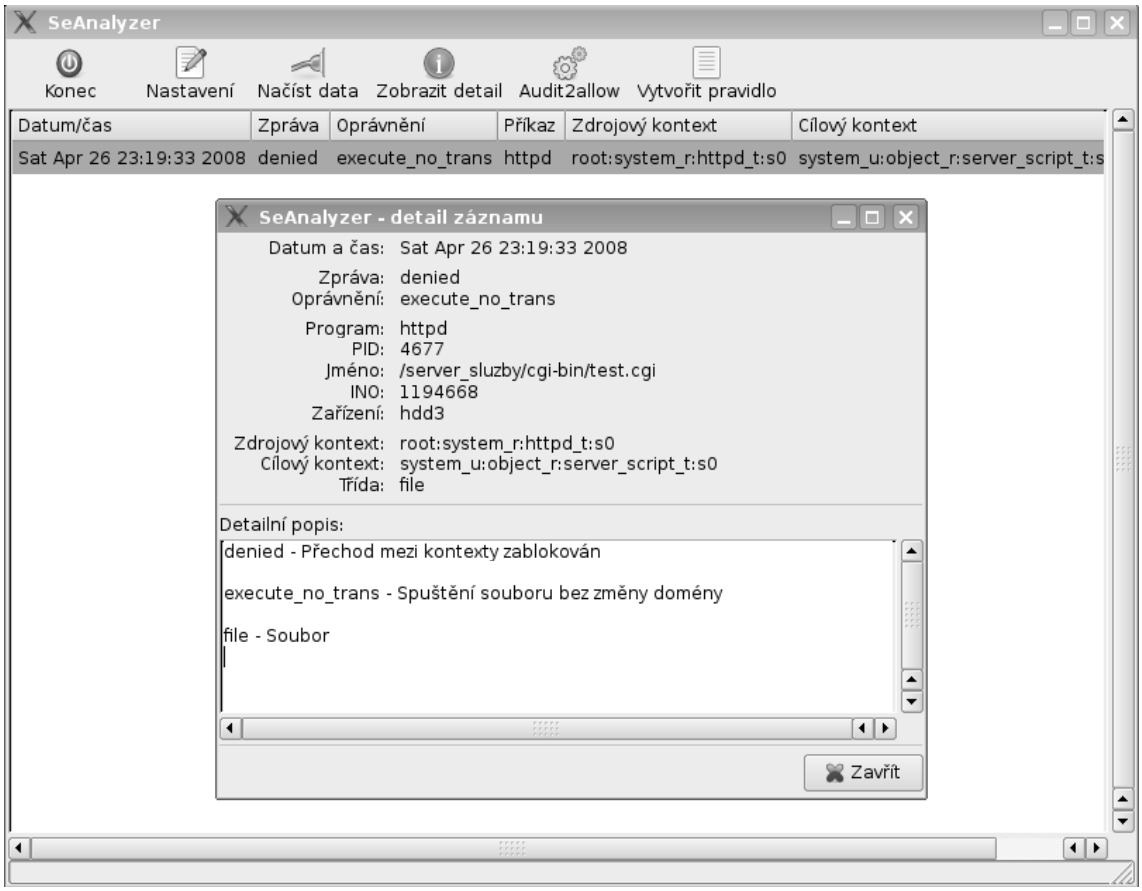
Souboru je potřeba nastavit práva spouštění pomocí `chmod +x text.cgi`. Dále je nutné nastavit v definici virtuálních domén, odkud mají brát CGI skripty. Toto docílíme přidáním tohoto řádku do definice jednotlivých domén a restartováním httpd serveru.

```
ScriptAlias /cgi-bin/ /server_sluzby/cgi-bin/
```

Nyní je vše připraveno a my se můžeme pokusit otevřít stránku s CGI skriptem – to lze například pomocí pokusu o otevření `http://domena1.cz/cgi-bin/test.cgi`. Podle úvahy bychom předpokládali, že máme povoleno vše pro to, aby skripty fungovaly. Avšak dostaneme pouze hlášku o špatné konfiguraci serveru. Proto musíme přistoupit k úpravě pravidla.

Úprava modulu

Abychom věděli, co máme do definice přidat, nejdříve je potřeba načtením záznamů zjistit, co je ještě zakázáno – výpis je vidět na obrázku 6.6 a lze z něj zjistit, že je pro souborovou třídu zablokována operace *execute_no_trans*. Proto je potřeba její povolení dopsat do pravidel.



Obr. 6.6: Záznam po pokusu o načtení CGI skriptu

Otevřeme si znovu okno pro manuální psaní, a jak jsme již dělali, vyplníme název adresáře a jméno modulu a pomocí tlačítka načteme data ze serveru a vyplníme příslušná pole formuláře. Budeme upravovat pouze soubor TE a doplníme pravidlo pro povolení zablokované operace:

```
allow httpd_t server_script_t:file { execute_no_trans };
```

Povolovanou operaci lze samozřejmě doplnit k již definovaným povolením pro třídu souborů. Nyní je potřeba opět projít přes sestavení a nahrání modulu do jádra SELinuxu. Přeznačení souborového systému není potřeba, protože již nezasahujeme do značení souborů.

Nyní je nutno opět již známým stylem pokus-omyl vyzkoušet, zda je současná definice dostačující nebo je potřeba ještě něco povolit. Otevřeme tedy znovu již zkoušenou stránku s našim testovacím CGI skriptem. Měly bychom se dobrat zdárného konce, protože námi očekávaná stránka je zobrazena. Pro jistotu ještě načteme záznamy SELinuxu. Zjistíme, že se již žádná další výjimka neobjevila, tudíž se nám podařilo i základní zabezpečení CGI skriptů pomocí samostatného typu.

7 ZADÁNÍ PRO POČÍTAČOVÉ CVIČENÍ

V akademickém roce 2007/2008 je počítačové cvičení předmětu „Správa, návrh a bezpečnost počítačových sítí“ vyučováno dvěma studenty doktorského studia, přičemž každý z nich vede výuku trochu odlišně a hlavně používá jinou distribuci Linuxu – využívají se distribuce Debian a CentOS. Také proto program vzniká tak, aby byl využitelný právě v těchto dvou distribucích. Navrhnout přesné znění zadání pro cvičení není úplně jednoduché a nelze jej zařadit libovolně v semestru. Proto by bylo dobré zde pouze nastínit, jak by mohlo být cvičení zaměřené na výuku řízení přístupu MAC vedeno. Přesné vysvětlení a vytvoření přesných úkolů pro počítačové cvičení by bylo vhodnější nechat až na příslušného vedoucího počítačového cvičení.

Celé cvičení bychom mohli rozdělit v podstatě na 4 části:

1. Vysvětlení základního principu řízení přístupu v Linuxových systémech, uvedení nedostatků tohoto řešení. Úvod do povinného řízení přístupu a základní popis jednotlivých technologií (AppArmor, GrSecurity, SELinux). Bližší seznámení se systémem SELinux.

2. Vysvětlení základního principu psaní pravidel – zejména definice TypeEnforcement, ale také další soubory sloužící pro definici pravidel (.fc, .if). Ukázka instalace vývojových nástrojů pro tvorbu nových pravidel a základní ukázka použití utilit, které se používají při tvorbě – audit2allow, checkmodule, semodule, fixfiles.

3. Ukázka základní práce s aplikací, která vznikla v této diplomové práci. Předpokladem bude její přítomnost v hostitelském systému a dostatečná práva na spuštění. Práce s SSH klíči by měla být probrána v předchozích cvičeních, tudíž by studenti neměli být tímto zaskočeni.

4. Zadání úkolů na samostatnou práci – jako doporučení bych vycházel z kapitoly 6, protože dle mého názoru jde o poměrně jednoduchou ukázku základního zabezpečení běžného webového serveru i s omezením spouštění CGI skriptů pouze na určitý adresář.

Pro praktickou ukázku a práci studentů by byla určitě vhodnější distribuce CentOS, protože je do ní technologie SELinux již od instalace pevně zakomponována. V distribuci Debian přece jen nejde o primární součást a je potřeba ji doinstalovat a povolit i v zavaděči systému. Proto bych doporučil třeba pouze pro toto cvičení využít přednastavený virtuální počítač právě s distribucí CentOS.

8 ZÁVĚR

Technologie SELinux je velmi rozsáhlým projektem, který se každým dnem značně vyvíjí. Vzhledem ke své komplexnosti je možné díky ní zajistit takovou bezpečnost, která by měla být dostačující každému správci serverů. Problémem však může být nutnost počátečního studia, protože pochopení není vůbec jednoduché. Ale vzhledem k tomu, že jsou nyní již velmi dobře definována pravidla pro většinu používaných síťových serverů, k samotné definici nových pravidel správce serveru tak často nedostane. Přesto by ale bylo dobré, kdyby každý, kdo pracuje se systémem Linux měl aspoň základní povědomí o tom, co je to povinné řízení přístupu a uměl definici aspoň základních pravidel u jeho nejrozsáhlejšího projektu – SELinuxu.

Cílem diplomové práce bylo navrhnout a implementovat aplikaci, která by sloužila pro analýzu záznamů SELinuxu a umožňovala základní psaní nových pravidel.

Při zpracování jsem zvolil vytvoření grafické aplikace, která bude pracovat na vzdáleném serveru pomocí protokolu SSH – grafický vzhled je zvolen s ohledem na uživatelskou přívětivost, protože je počítáno i s využitím programu při výuce předmětu „Správa, návrh a bezpečnost počítačových sítí“.

V aplikaci je využita plně i poloautomatická tvorba pravidel pomocí utility Audit2allow a samozřejmě také možnost vytváření nových pravidel manuálně. To je umožněno pomocí jednoduchého formuláře, který by měl za pomoci záložek vést uživatele krok po kroku tak, aby byl schopen zapsat nová pravidla.

V poslední části práce jsem navrhl pro vyučující počítačového cvičení obecný návrh toho, jak by měla být zhruba členěna vyučovací hodina, která je určena k výuce SELinuxu. Doporučil jsem také použití distribuce CentOS, i když je aplikace psána tak, že je počítáno i s použitím linuxové distribuce Debian.

LITERATURA

[1] MARŤÁK, Pavel. Bezpečnost dat v praxi. *IT Systems* [online]. 2005, č. 4 [cit. 2007-10-23]. Dostupný z WWW: <http://www.systemonline.cz/clanky/bezpecnost-dat-v-praxi.htm>. ISSN 1802-615X.

[2] MCCARTY, Bill. *SELinux : NSA's Open Source Security Enhanced Linux*. Sebastopol (CA) : O'Reilly, 2004. 254 s. ISBN 0-596-00716-7.

[3] MAYER, Frank, MACMILLAN, Karl, CAPLAN, David. *SELinux by Example : Using Security Enhanced Linux*. Crawfordsville (Indiana) : Prentice Hall, 2006. 456 s. ISBN 0-131-96369-4.

[4] *National Security Agency : Security-Enhanced Linux* [online]. 2007 [cit. 2007-11-16]. Dostupný z WWW: <http://www.nsa.gov/selinux/>.

[5] HOLAS, Martin. *Povinné řízení přístupu* [online]. 2005. Brno : [2005] [cit. 2007-10-31]. Dostupný z WWW: <http://www.fi.muni.cz/∼kas/p090/referaty/2005-podzim/st/selinux.html>.

[6] PALÁT, Pavel. Grsecurity. *Linuxzone* [online]. 2004 [cit. 2007-10-31]. Dostupný z WWW: <http://www.linuxzone.cz/index.phtml?ids=1&idc=1060>. ISSN 1213-8738.

[7] *Red Hat Enterprise Linux 5 Documentation* [online]. c2007 [cit. 2007-10-29]. Dostupný z WWW: <http://www.redhat.com/docs/manuals/enterprise/RHEL-5-manual/>.

[8] *SELinux - Fedora Project Wiki* [online]. 2008-05-04 [cit. 2008-04-30]. Dostupný z WWW: <http://fedoraproject.org/wiki/SELinux>.

[9] *AppArmor Application Security for Linux* [online]. c2008 [cit. 2007-10-29]. Dostupný z WWW: <http://www.novell.com/linux/security/apparmor/>.

[10] CURPHEY, Mark, et al. *A Guide to Building Secure Web Applications* [online]. c2002 , Sun Sep 22 2002 [cit. 2007-11-10]. Dostupný z WWW: <http://www.cgisecurity.com/owasp/html/index.html>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

API Application Programming Interface – Rozhranní pro programování aplikací

CPL Current Privillege Level

CPU Central Processing Unit – Procesor

DAC Discretionary access control – Volitelné řízení přístupu

MAC Mandatory access control – Povinné řízení přístupu

MLS Multilevel security – Vícevrstvá bezpečnost

PaX Záplata linuxového jádra

RBAC Role-based access control – Řízení přístupu založené na rolích

TE Type Enforcement – Vynucení typu

TLB Translation Lookaside Buffer

SEZNAM PŘÍLOH

A OPRÁVNĚNÍ SELINUXU

allow:	Přístup k tomuto kontextu povolen
denied:	Přechod mezi kontexty zablokován

B SEZNAM MOŽNÝCH VÝJIMEK

accept:	Přijetí spojení
acceptfrom:	Přijetí spojení od klientského soketu
add_name:	Přiřazení jména
append:	Zápis nebo připojení obsahu souboru či soketu
associate:	Přiřazení souboru s klíčem k systému souborů, frontě, semaforu či segmentu paměti
avc_toggle:	Přepnutí mezi permissive (tolerantní) a enforcing (prosazovací) módem
bdflush:	Správa démona buffer-dirty-flush
bind:	Přiřazení jména soketu
change_sid:	Určení SID objektu během přeznačování
check_context:	Zapsání kontextu do selinuxfs souborového systému
chfn:	Změna informací uživatelského účtu (jméno, kancelář, telefon, ...)
chown:	Změna vlastníka souboru a změna skupinové vlastnictví
chsh:	Změna přihlašovacího shellu
compute_av:	Vypočítání přístupového vektoru, který je dán zdrojem, cílem a třídou.
compute_create:	Zapsání informace o vytvoření do selinuxfs souborového systému
compute_member:	Stanovení informace o členu do selinuxfs souborového systému
compute_relabel:	Nastavení informace o přeznačení do selinuxfs souborového systému
compute_user:	Nastavení informace o uživateli do selinuxfs souborového systému
connect:	Vytvoření spojení
connectto:	Připojení na soket serveru
context_to_sid:	Převod kontextu na SID
create:	Vytvoření nového souboru, IPC objektu, fronty, semaforu nebo segmentu sdílené paměti
dac_override:	Obejití volitelného řízení přístupu kromě LINUX_IMMUTABLE
dac_read_search:	Obejití všech volitelných řízení přístupu
destroy:	Odstranění IPC objektu, fronty zpráv, nastavení semaforu nebo segmentu sdílené paměti
enforce_dest:	Cílový uzel může vnutit oprávnění na cílový soket
enqueue:	Zpráva, která může být ve frontě
entrypoint:	Vstup do nové domény přes tento program
execute:	Spuštění
execute_no_trans:	Spuštění souboru bez změny domény
fork:	Větvení na 2 procesy
fowner:	Povolit operace nad soubory, které jsou rozdílné od těch, které jsou potlačeny vlastnictvím
fsetid:	Obejít kontrolu efektivního ID uživatele s ohledem na nastavení ID uživatele a nastavení ID skupiny na soubory
get_sid:	Získání seznamu aktivních SID
get_user_sid:	Získání seznamu aktivních SID
getattr:	Získání vlastností souborů, procesů, front zpráv nebo segmentů sdílené paměti
getcap:	Získání schopností procesu
getopt:	Získání vlastností soketu
getpgid:	Získání skupinového ID procesu
getsched:	Získání priority procesu
getsession:	Získání ID sezení
ioctl:	Požadavky na řídící systémová volání I/O, která nejsou vyvolána dalšími oprávněními
ipc_info:	Získání informace pro IPC soket
ipc_lock:	Zamčení sdílených a nesdílených paměťových segmentů
ipc_owner:	Ignorování kontroly vlastnictví IPC
kill:	Vyvolání signálu na jakémkoliv procesu
lease:	Získání pronájmů funkce fnctl na souboru
link:	Vytvoření tvrdého odkazu na soubor
linux_immutable:	Změna S_IMMUTABLE a S_APPEND atributů souboru na souborových systémech,

	které to podporují
listen:	Naslouchání příchozích spojení
load_policy:	Načtení bezpečnostních pravidel
lock:	Nastavit či zrušit zamknutí souboru či paměťových stránek
member_sid:	Získání SID pro výběru člena objektu s více instancemi
mkmod:	Vytvoření uzlů znakových nebo blokových zařízení
mount:	Připojení souborového systému
mounton:	Použití jako přípojného bodu souborového systému
name_bind:	Svázání portu s IP nebo souboru s Unixovým soketem
net_admin:	Změna síťového nastavení
net_bind_service:	Spojení na privilegovaném portu
net_raw:	Otevření normálního či paketového soketu
netbroadcast:	Poslad síťový broadcast nebo naslouchat příchozím multicast zprávám
newconn:	Vytvoření nového soketu pro spojení
nfsd_control:	Správa NFS serveru
noatsecure:	Povolení bezpečnostního módu GLibc
node_bind:	Svázání soketu
passwd:	Změna uživatelského hesla
ptrace:	Sledovat provádění program rodičovského či děděného procesu
quotaget:	Získání informací o kvótách
quotamod:	Změna informací o kvótách
quotaon:	Povolení kvót
rawip_recv:	Příjem IP paketu
rawip_send:	Poslání IP paketu
read:	Čtení obsahu osuboru, IPC, fronty zpráv nebo segmentu sdílené paměti
receive:	Odstranění zprávy z fronty
recv_msg:	Přijetí datagramové zprávy, která má jiné SID než soket
recvfrom:	Příjem datagramu ze soketu
relabelfrom:	Změna bezpečnostního kontextu založená na existujícím typu
relabelto:	Změna bezpečnostního kontextu založená na novém typu
remount:	Změna vlastnostní připojeného souborového systému
remove_name:	Odstranění jména
rename:	Přejmenování tvrdého odkazu
reparent:	Změna mateřského adresáře
rlimitinh:	Zdědění omezení prostředků ze starého SID
rmdir:	Vymazání adresáře
rootok:	Změna hesla v případě uživatele root a pokud má proces oprávnění root
search:	Prohledání adresáře
send:	Přidání zprávy do fronty
send_mng:	Poslání datagramové zprávy s rozdílným SID než soketu, na který je posílána
sendto:	Poslání datagramu na soket
setattr:	Změnit atributy souboru, sdíleného segmentu paměti nebo fronty zpráv
setbool:	Nastavit logické proměnné
setcap:	Nastavit schopnosti procesů
setenforce:	Změnit vynucovacího módu SELinuxu
setfscreate:	Nastavit kontext operace fscreate
setgid:	Povolit volání operace setgig a padělání ID skupiny na doporučení přicházející ze soketu
setopt:	Nastavit vlastnosti IPSec nebo soketu
setpcap:	Přenést mapu schopností procesu
setpgid:	Nastavení skupinového ID procesu
setrlimit:	Změna tvrdých limitů procesu
setsched:	Nastavení priority procesu
setuid:	Povolit operaci setuid a padělat UID na doporučení přicházející ze soketu
share:	Povolit sdílení stavu mezi klonovaným či větveným procesem
shutdown:	Zastavit spojení
sid_to_context:	Změnit SID na kontext
sigchld:	Poslání SIGCHLD signálu
siginh:	Zdědit stav signálu ze starého SID

sigkill:	Poslání SIGKILL signálu
signal:	Poslání jiného signálu než SIGKILL, SIGSTOP nebo SIGCHLD
signull:	Test na existenci dalšího procesu bez poslání signálu
sigstop:	Poslání SIGSTOP signálu
swapon:	Povolení souboru pro použití jako swap
sys_admin:	Různé systémové schopnosti
sys_boot:	Restart systému
sys_chroot:	Použití chroot prostředí
sys_module:	Načtení či odstranění modulů jádra Linuxu
sys_nice:	Změnit prioritu procesu
sys_pacct:	Změnit stav vykazování procesu
sys_ptrace:	Stopování procesu
sys_rawio:	Vykonání I/O opearace (čtení/zápis)
sys_resource:	Různé schopnosti
sys_time:	Nastavení systémového času a hodin reálného času
sys_tty_config:	Nastavení tty zařízení
syslog_console:	Zaznamenání do konzole syslogu
syslog_mod:	Vykonání operace nad syslogem, avšak jiné než čtení či zaznamenávání do konzole
syslog_read:	Čtení syslogu
tcp_recv:	Přijetí TCP paketu
tcp_send:	Poslání TCP paketu
transition:	Přechod na nový SID
transition_sid:	Určení SID pro nový objekt
udp_recv:	Přijetí UDP paketu
udp_send:	Poslání UDP paketu
unix_read:	Vykonání čtení IPC
unix_write:	Vykonání zápisu či doplnění IPC
unlink:	Odstranění tvrdého odkazu
unmount:	Odpojení souborového systému
use:	Použití původního dekriptoru souboru
write:	Zapsat nebo doplnit obsah objektu IPC nebo souboru

C TŘÍDY OBJEKTŮ

blk_file:	Blokové zařízení
chr_file:	Znakové zařízení
dir:	Adresář
fd:	Souborový popisník
fifo_file:	Soubor FIFO (First In First Out)
file:	Soubor
filesystem:	Naformátovaný souborový systém patřící dané partišně
lnk_file:	Odkaz či symbolický odkaz
sock_file:	Soubor síťového soketu
ipc:	IPC
msg:	Komunikační meziprocesorová zpráva ve frontě
msgq:	Meziprocesorová komunikační fronta
sem:	Meziprocesorový komunikační semafor
shm:	Sdílená paměť meziprocesorové komunikace
key_socket:	IPSec soket
netif:	Síťové zařízení
netlink_socket:	Soket využitý pro komunikaci s kernelem za pomoci volání netlinku
netlink_selinux_socket:	Soket využitý pro komunikaci s jádrem SELinuxu
node:	TCP/IP host reprezentovaný IP adresou
rawip_socket:	IP soket
socket:	Defaultní soket
tcp_socket:	TCP soket
udp_socket:	UDP soket
unix_dgram_socket:	Datagramový soket v Unixových systémech
unix_stream_socket:	Vysílací soket v Unixových systémech
passwd:	Soubor s Linuxovými hesly
capability:	SELinuxová způsobilost
process:	Proces
security:	Objekty související s bezpečností (např. bezpečnostní politika SELinuxu)
system:	Objekty systému a jádra